

SHELLELASTIC: SCALABLE ELASTIC PRETRAINING WITH SIMULTANEOUS MAXIMAL UPDATES AND HYPERPARAMETER TRANSFER

Can Jin^{1*}, Yan Kang², Yifan Gong², Chongjian Ge², Mingze Xu², Yuanjun Xiong²,
Dimitris N. Metaxas¹, Hongwu Peng²

¹Rutgers University ²Adobe Research

🌐 Project Page

https://jincan333.github.io/ShellElastic_page

<> Code

<https://github.com/jincan333/ShellElastic>

📁 Checkpoints & Data

<https://huggingface.co/spaces/Can111/ShellElastic>

ABSTRACT

Foundation models need to support diverse inference budgets, yet training a separate model for each budget duplicates pretraining effort. Elastic models train several nested sub-networks inside one weight-shared supernet, but existing recipes face three obstacles: variable per-step workloads that hinder distributed scaling, limited hyperparameter (HP) transfer across model sizes, and no guarantee that all sub-networks are maximally updated at once. We present **SHELLELASTIC**, an elastic transformer that organizes hidden dimensions into a dense core and successive shells, so that every hidden matrix is block lower triangular. One full-width forward pass contains every sub-network’s exact hidden states as prefixes, so joint training keeps a fixed computation graph and costs about as much per step as training the widest sub-network alone. We further introduce μ -**ELASTIC**, an extension of maximal update parameterization (μ P) to **SHELLELASTIC** and other elastic architectures. Its scaling rules provably keep all sub-networks of a supernet maximally updated simultaneously (μ -Internal), and keep doing so when the supernet is widened or given more shells; sweeps confirm that HPs tuned on a small non-elastic proxy stay near-optimal for elastic families and larger supernets (μ -Transfer). A single **SHELLELASTIC** run produces sub-networks that outperform MatFormer and fully elastic baselines at matched parameter counts, and approach separately trained models that together take several times the steps.

1 INTRODUCTION

Foundation models need to support different compute budgets for different tasks and deployment platforms. Elastic training answers this need by optimizing several nested sub-networks inside one shared parameter set: slimmable networks and Once-for-All established the idea for vision models (Yu et al., 2018; Cai et al., 2020), MatFormer nests the FFN width of a transformer and underlies released on-device models (Devvrit et al., 2024; Gemma Team, 2025), and Flextron and Nemotron Elastic make pretrained LLMs elastic by post-training (Cai et al., 2024; Taghibakhshi et al., 2025).

Three problems remain when the elastic training recipe is carried into foundation-model pretraining.

Scalability. Standard recipes sample one or a few sub-networks per step (Devvrit et al., 2024; Yu & Huang, 2019), so the computation graph, the parameter access pattern and the workload change from step to step; under parameter sharding this variability complicates activation memory, recomputation and distributed scheduling, the very machinery that large-scale pretraining depends on. Training all sub-networks at every step fixes the workload but needs a forward and backward pass per sub-network, so the cost grows with the family size and erodes the saving that motivates elastic training.

*Corresponding author: can.jin@rutgers.edu. Work done during an internship at Adobe Research.

Transferability. Hyperparameters (HPs) tuned at one model size need not remain effective at another, and repeating the search over learning rate, initialization scale and weight decay at every target scale cancels the compute that elastic training saves. Maximal update parameterization transfers HPs across widths for independently trained models (Yang et al., 2021); elastic models add two questions: can HPs tuned on a non-elastic proxy serve an entire elastic family, and can those tuned on a smaller elastic family serve a larger one? We refer to these two forms of transfer as μ -Transfer for elastic models and seek one scaling rule that delivers both under parameter sharing.

Simultaneous maximal updates. Existing elastic training recipes do not generally ensure maximal updates for all sub-networks simultaneously.¹ Without width-dependent scaling, HPs that yield well-scaled feature updates in one sub-network make updates vanish or blow up in another, and re-tuning one global value only trades one width against another, so some widths stay mis-scaled. The requirement is therefore a shared parameterization that keeps every width maximally updated simultaneously despite parameter sharing; we call this property μ -Internal for elastic models.

We address the first problem with a new architecture, **SHELLELASTIC**, and the other two with μ -**ELASTIC**. **SHELLELASTIC** (Figure 1) partitions the residual stream into a dense core followed by ordered shells, each a multiple of a base width g , and each supported sub-network is the core plus a prefix of the shells. Every hidden matrix is block lower triangular: a shell reads the core, the earlier shells and itself, but never a later shell. Trunk normalization acts within each group, attention heads align with the group boundaries, and every supported width has its own readout. One full-width forward pass therefore contains the exact hidden states of every sub-network as prefixes (Proposition 1), and the per-step compute stays close to that of a single full-width pass however many sub-networks are trained, so every step has a fixed workload, as in non-elastic training.

μ -**ELASTIC** (Figure 2) extends maximal update parameterization (μ P) (Yang et al., 2021) to elastic models with two factors: a μ -Transfer factor $m = g/d_0$ that carries HPs from a non-elastic proxy of width d_0 to base width g , and a μ -Internal factor for the different widths inside one shared model. Every group reads the same input coordinates in every sub-network that contains it, so we set its initialization, learning rate and weight decay from this fan-in, while each readout uses its own input width (Table 1). This keeps every group maximally updated in all sub-networks at once (Theorem 1), which uniform scaling across groups cannot do in general (Proposition 2). The same two factors apply to MatFormer and to fully elastic models (Appendix D).

Our main contributions are summarized as follows:

- ★ **Scalable elastic architecture.** We propose **SHELLELASTIC**, an elastic transformer whose single full-width forward pass yields the exact hidden states of every sub-network as prefixes, so every step has a fixed workload rather than a varying one.
- ★ **μ -ELASTIC parameterization.** We introduce μ -**ELASTIC**, which transfers HPs from a non-elastic proxy to an elastic model and from smaller to larger elastic models (μ -Transfer), and keeps all sub-networks of a supernet maximally updated at once (μ -Internal).
- ★ **Theoretical and empirical validation.** We prove that μ -**ELASTIC** keeps all sub-networks maximally updated at both levels of transfer; sweeps confirm that transferred HPs stay near-optimal and that μ -**ELASTIC** beats the standard parameterization on every elastic family.
- ★ **Comprehensive ablation studies.** We search architectures at equal parameters and ablate sub-network selection, loss weights, normalization scope, readout design, in-place distillation, sub-network ensembles and stage-residual gradient scaling for elastic pretraining.

2 RELATED WORK

Elastic models. Elastic models train several nested sub-networks in one shared parameter set, so one run serves many inference budgets (Yu et al., 2018; Cai et al., 2020; Devvrit et al., 2024). Slimmable networks train one convolutional network at several widths (Yu et al., 2018; Yu & Huang, 2019), Once-for-All also nests depth, kernel size and resolution (Cai et al., 2020), and the same idea has been applied to the width and depth of BERT (Devlin et al., 2019; Hou et al., 2020) and to output

¹A parameter group is *maximally updated* when the pre-activations it produces are $\Theta(1)$ at initialization and move by $\Theta(\eta_0)$ after one optimizer step, with η_0 the base learning rate before width scaling, so that its contribution neither vanishes nor blows up as the width grows (Yang & Hu, 2021; Yang et al., 2021).

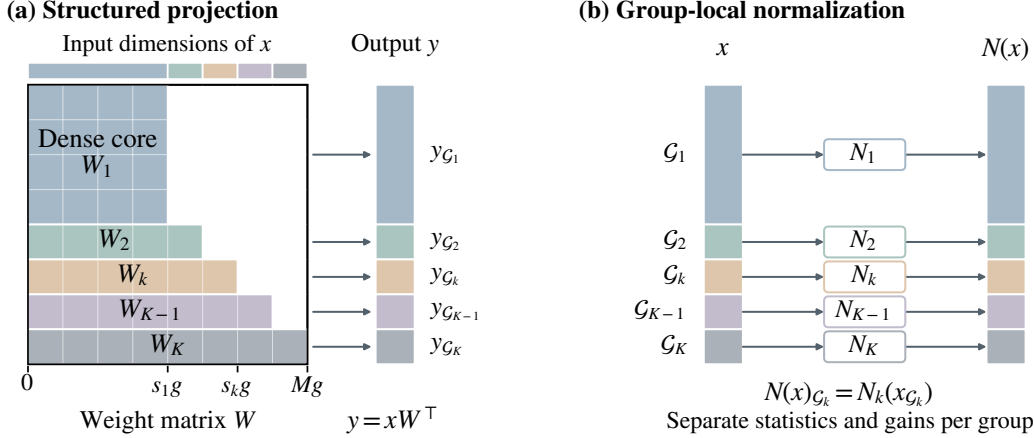


Figure 1: **Core and shells of SHELLELASTIC preserve supported prefixes.** (a) Each colored block row produces output group $\mathcal{G}_k$ from the input prefix ending at $s_k g$. Group $\mathcal{G}_1$ is the dense core; subsequent groups are shells. (b) N_k normalizes group $\mathcal{G}_k$ using its own statistics.

representations (Kusupati et al., 2022); these recipes train the sub-networks jointly or by sampling, usually with distillation from the largest one (Hinton et al., 2015). For decoder-only transformers (Vaswani et al., 2017), MatFormer (Devvrit et al., 2024) nests only the FFN hidden width and underlies released on-device models (Gemma Team, 2025), while Flextron and Nemotron Elastic extract nested sub-networks from a pretrained model (Cai et al., 2024; Taghibakhshi et al., 2025). The concurrent StairFormer (Trost et al., 2026) nests every smaller model as a prefix of one forward pass and instantiates this nesting with equal-width blocks; in its experiments, it trails dense models of the same size at every budget but supports a wider range of usable budgets than MatFormer. **SHELLELASTIC** uses the same prefix structure but chooses the supported widths, which set the core and shell sizes, and normalizes within groups instead of over cumulative prefixes. No existing work derives how HPs should scale across nested widths or keeps every sub-network maximally updated.

HP transfer and μ -parameterization. Tuning HPs such as the learning rate and initialization is expensive at scale, and under the standard parameterization (Sohl-Dickstein et al., 2020) their optima drift with width, so a small model cannot stand in for a large one (Yang et al., 2021). Scaling HPs with model size removes this drift, so values tuned on a small proxy transfer zero-shot (Yang et al., 2021; Dey et al., 2026). Building on infinite-width theory (Jacot et al., 2018; Mei et al., 2018), Yang & Hu (2021) identified μ P, a parameterization that gives every layer maximal $\Theta(1)$ feature updates as width grows, and Yang et al. (2021) turned it into a transfer rule for initialization, learning rate and multipliers. CompleteP extends the rule across depth for pre-norm transformers (Yang et al., 2024; Bordelon et al., 2024; Dey et al., 2026), and Mlodozieniec et al. (2026) add batch size and token budget. Everett et al. (2024) derive per-layer learning-rate exponents across parameterizations and optimizers, including Adam, and find that HP transfer is not limited to μ P; Table 1 follows their scaling rules. These rules assume separately trained models, where each tensor has one fan-in; **μ -ELASTIC** extends them to the shared tensors of an elastic supernet.

3 METHOD

In the **SHELLELASTIC** architecture, all sub-networks reuse hidden-state prefixes from one full-width trunk pass; **μ -ELASTIC**, a parameterization based on each weight’s active input width, enables reuse of base optimization settings across model sizes.

3.1 THE SHELLELASTIC ARCHITECTURE

Core, shells, and supported widths. We organize hidden dimensions into a dense *core* followed by successive *shells* (Figure 1). For base width g , choose supported sub-network width indices

$$\mathcal{S} = \{s_1 < \dots < s_K = M\} \subseteq \{1, \dots, M\}, \quad D_{\text{model}}^{(s)} = sg, \quad D_{\text{model}} = Mg. \quad (1)$$

With $s_0 = 0$, these boundaries define contiguous groups $\mathcal{G}_k = \{s_{k-1}g + 1, \dots, s_k g\}$ of size $(s_k - s_{k-1})g$. The core is $\mathcal{G}_1$, and the shells are $\mathcal{G}_2, \dots, \mathcal{G}_K$. Sub-network s_k contains the first k groups.

Table 1: **μ -ELASTIC for SHELLELASTIC.** Scaling rules set by the μ -Transfer factor m , the μ -Internal factor s (Equation 7), and the FFN expansion $e = D_{\text{ffn}}/D_{\text{model}}$; s is the output group’s supported boundary for hidden weights and the sub-network width index for readouts. Initialization gives nominal weight standard deviations, or initial values for gains and scalars.

Parameter	Active input width	Initialization	Learning rate	Weight decay	Multiplier
Token embedding E	V (fixed)	σ_0	η_0	λ_0	1
Attention; FFN up/gate	sg	$\sigma_0/\sqrt{ms}$	$\eta_0/(ms)$	$\lambda_0 ms$	1
FFN down	esg	$\sigma_0\sqrt{e}/\sqrt{ms}$	$\eta_0/(ms)$	$\lambda_0 ms$	$1/e$
Readout U_s	sg	$\sigma_0/\sqrt{ms}$	$\eta_0/(ms)$	$\lambda_0 ms$	α_s
Norm gains; readout scalar α_s	1 (elementwise)	1	η_0	0	1

Projections and normalization. Write a token’s hidden state as a row vector x , with $x_{:d}$ denoting its first d dimensions. A square projection is $y = xW^\top$, where $W \in \mathbb{R}^{D_{\text{model}} \times D_{\text{model}}}$. For output group k , the block row $W_k \in \mathbb{R}^{|\mathcal{G}_k| \times s_k g}$ reads only the first $s_k g$ input dimensions:

$$y_{\mathcal{G}_k} = x_{:s_k g} W_k^\top. \quad (2)$$

This makes W block lower triangular at the group boundaries (Figure 1a).

Group-local normalization uses a separate operator N_k and statistics within each group (Figure 1b). For RMSNorm (Zhang & Sennrich, 2019):

$$[N_k(x_{\mathcal{G}_k})]_j = a_j \frac{x_j}{\sqrt{|\mathcal{G}_k|^{-1} \sum_{u \in \mathcal{G}_k} x_u^2 + \epsilon_N}}, \quad j \in \mathcal{G}_k. \quad (3)$$

Here a_j is a learned gain and $\epsilon_N > 0$ is a numerical stabilizer. The combined normalization satisfies $N(x)_{\mathcal{G}_k} = N_k(x_{\mathcal{G}_k})$. Both operations preserve prefixes ending at the boundaries in $\mathcal{S}$. Cuts within a group generally change the retained outputs, so sub-networks can be extracted only at these boundaries. Thus, $\mathcal{S}$ determines the block-triangular pattern of every projection, the normalization groups, and the available sub-network widths.

Attention and FFN. Both branches (Figure 7 in Appendix A) use the group-local normalization of Equation 3 as their pre-norm. Attention is elastic in its number of heads: sub-network s uses the first $D_{\text{model}}^{(s)}/d_{\text{head}} = sg/d_{\text{head}}$ heads, keeping the head dimension d_{head} fixed. We require g to be a multiple of d_{head} so that supported widths retain complete heads. The query, key, value, gate, and output projections follow Equation 2; they constrain dependencies among hidden dimensions before and after attention. Query/key normalization and rotary position transformations (Su et al., 2024) act within each head. Each active head a performs causal self-attention:

$$\text{Attn}(x) = \left[\text{concat}_a \left(\text{softmax} \left(Q_a K_a^\top / \sqrt{d_{\text{head}}} + C \right) V_a \right) \odot \text{sigmoid}(x W_G^\top) \right] W_O^\top, \quad (4)$$

where C is the causal mask, Q_a, K_a, V_a come from the corresponding projections of x and concatenation runs over the active heads. Adding heads leaves earlier heads’ attention outputs unchanged; the structured output projection then preserves the supported hidden-state prefixes.

We use a SwiGLU FFN (Shazeer, 2020) with expansion $e = D_{\text{ffn}}/D_{\text{model}}$ and integer base FFN width eg , so sub-network s has $D_{\text{ffn}}^{(s)} = e D_{\text{model}}^{(s)}$. All three maps use the same group dependency pattern. The up and gate weights have shape $D_{\text{ffn}} \times D_{\text{model}}$, with group boundaries $s_k g$ on the input axis and $es_k g$ on the output axis. The down weight has shape $D_{\text{model}} \times D_{\text{ffn}}$. The FFN computes

$$\text{FFN}(x) = [\text{SiLU}(x W_1^\top) \odot (x W_3^\top)] W_2^\top. \quad (5)$$

Elementwise operations and residual additions preserve the same prefixes.

Embedding and readout. Sub-network s uses sg -dimensional token embeddings and computes its logits from the first sg dimensions of the L -layer trunk output $x^{(L)}$:

$$z_s = \alpha_s N_s^{\text{out}}(x_{:sg}^{(L)}) U_s^\top, \quad s \in \mathcal{S}. \quad (6)$$

Here α_s is a learned logit scale, N_s^{out} normalizes over all sg selected dimensions, and $U_s \in \mathbb{R}^{V \times sg}$ is the readout matrix for a vocabulary of size V . Figure 6 in Appendix A shows the complete model.

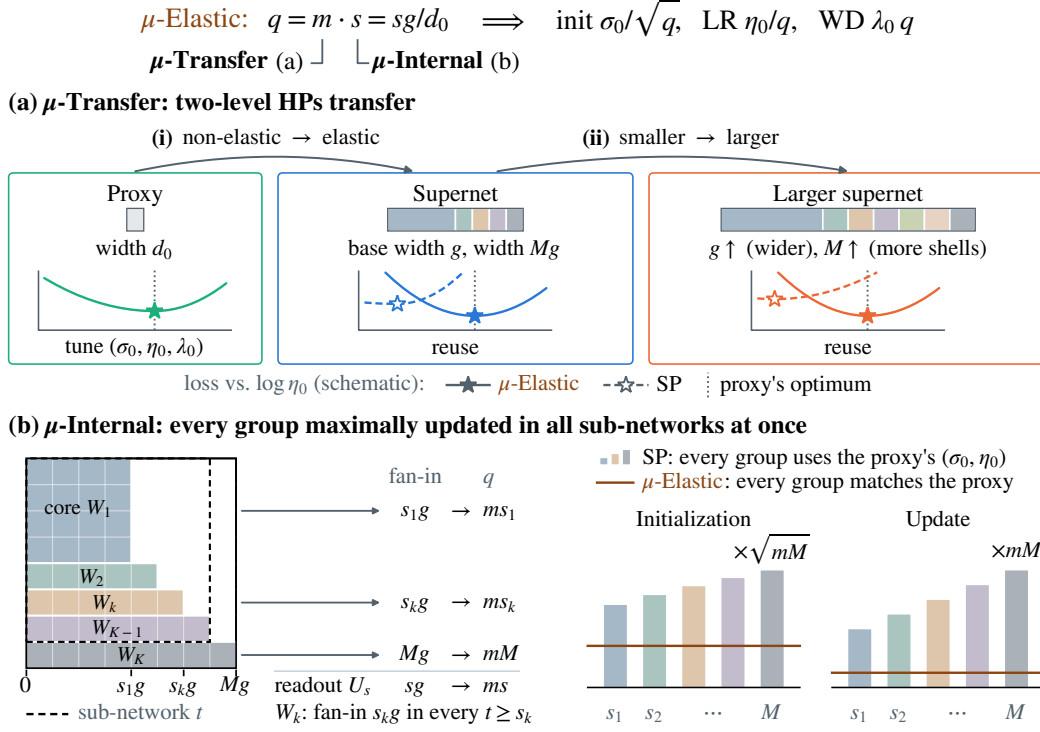


Figure 2: **Overview of μ -ELASTIC.** The ratio $q = m \cdot s$ of Equation 7 scales the HPs of every group and readout. (a) μ -Transfer: the $(\sigma_0, \eta_0, \lambda_0)$ tuned on the proxy are reused (i) for an elastic supernet and (ii) for a wider one with more shells (Corollary 1). (b) μ -Internal: block row W_k has fan-in $s_k g$ in every sub-network that contains it, so $q = ms_k$ keeps it maximally updated in all of them (Theorem 1). The bars show each group’s pre-activation size at initialization and update size: with the proxy’s (σ_0, η_0) , as under SP, all groups, even the core, exceed the proxy by $\sqrt{q}$ and q , so no shared value fixes them all (Proposition 2); under μ -ELASTIC, all match it.

Proposition 1 (Exact reuse at supported widths). *For fixed model parameters, let E_s be the first sg columns of the full-width embedding table, T_s the L -layer trunk restricted to width sg , and P_s the operator selecting the first sg hidden dimensions. For any token sequence u , evaluating sub-network s directly gives the same hidden states as extracting its prefix from sub-network t :*

$$T_s(E_s[u]) = P_s(T_t(E_t[u])), \quad s, t \in \mathcal{S}, \quad s \leq t.$$

Applying sub-network s ’s normalization and readout (Equation 6) to either result gives identical logits. Thus, one full-width trunk evaluation yields the same joint loss and parameter gradients as separate evaluations of all supported sub-networks. Appendix B proves the result by composing the prefix-preserving trunk operations across the L layers.

3.2 μ -ELASTIC: PARAMETERIZING ELASTIC MODELS

We introduce μ -ELASTIC to address the two optimization requirements (Figure 2): transferring HPs from a non-elastic proxy to an elastic family and between elastic model sizes (μ -Transfer), and maintaining maximal updates across sub-networks within each shared model (μ -Internal).

Composing transfer and internal scaling. Let a non-elastic proxy of width d_0 supply the base initialization scale σ_0 , base learning rate η_0 , and weight decay λ_0 . For base width g , define

$$m = \frac{g}{d_0}, \quad q(s) = \underbrace{m}_{\mu\text{-Transfer}} \cdot \underbrace{s}_{\mu\text{-Internal}} = \frac{sg}{d_0}, \quad s \in \mathcal{S}. \quad (7)$$

For a hidden weight, s is the supported boundary of its output group, whose input width stays sg (esg for an FFN down projection) in every sub-network that contains it, so one rule based on $q(s)$ serves all of them. The readout of sub-network s uses the same ratio.

Parameterization and optimization rules. Table 1 summarizes initialization, learning rate, weight decay, and forward multipliers by active input width for AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2019), following the per-layer scaling of Everett et al. (2024). Width is the only elastic axis, since all sub-networks share the depth L , minibatches and number of training steps; these shared quantities follow existing scaling rules for depth (Dey et al., 2026), batch size and token budget (Mlodozienec et al., 2026). The learning rate and weight decay scalings cancel, so AdamW applies the same relative decay to every decayed parameter group. The embedding is a lookup over a fixed vocabulary of size V , so it uses the base $(\sigma_0, \eta_0, \lambda_0)$. For FFN down projections, the $\sqrt{e}$ initialization factor and $1/e$ forward multiplier account for the expanded input width esg . The learned scalars α_s in Equation 6 adjust logit coefficients beyond width scaling. Norm gains and α_s are initialized to one and use learning rate η_0 with zero weight decay. The *standard parameterization* (SP) sets $m = s = e = 1$ only in the scaling rules and fixes $\alpha_s = 1$, preserving the architecture.

Theoretical guarantees. The statements below are proved in Appendix C under the standard heuristic assumptions of μ P (Yang et al., 2021): before the scaling of Table 1 is applied, the pre-activations of a group with active input width sg are $\Theta(\sqrt{sg})$ at initialization and move by $\Theta(sg)$ after one step, which is why the initialization and the learning rate carry different exponents.

Theorem 1 (μ -Internal: simultaneous maximal updates). *Under the assumptions of Appendix C and the rule of Table 1, consider a hidden group with supported boundary s and any sub-network $t \in \mathcal{S}$ with $t \geq s$. In every layer, its pre-activations inside sub-network t (i) have $\Theta(1)$ coordinates at initialization and (ii) move by $\Theta(\eta_0)$ after one AdamW step on that group, with constants independent of s , t , M , and m . The logits of every readout U_s have the same two properties.*

Equation 7 is what makes this possible: a group with boundary s sums over exactly sg input coordinates in every sub-network containing it, so one initialization scale and one learning rate satisfy both conditions for all $t \geq s$ at once. Training several sub-networks jointly does not change this, since they all feed a group the same input and Adam is invariant to the gradient scale.

Proposition 2 (A shared rule leaves groups mismatched). *If all groups of a hidden matrix share one initialization variance, learning rate and multiplier, the groups with boundaries $s < s'$ differ by factors $\Theta(\sqrt{s'/s})$ in pre-activation size and $\Theta(s'/s)$ in update size, whatever the shared values. For unbounded M/s_1 , no shared rule thus keeps every group maximal in the sense of Theorem 1.*

Under SP, the groups of a hidden matrix share (σ_0, η_0) , so by Proposition 2 re-tuning only trades the core against the outermost group, which stay $\sqrt{M/s_1}$ and M/s_1 apart in initialization and update; the best HPs are compromises depending on m , M and $\mathcal{S}$. Figure 3 and Table 3 show both effects: the SP optimum departs from the proxy’s, and SP trails μ -ELASTIC even after re-tuning.

Corollary 1 (μ -Transfer: two-level transfer). *Theorem 1 holds for every base width $g = md_0$ and every supported set $\mathcal{S} \subseteq \{1, \dots, M\}$. Hence the values $(\sigma_0, \eta_0, \lambda_0)$ tuned on the width- d_0 proxy (i) keep every group and readout of a SHELLELASTIC supernet maximally updated, and (ii) keep doing so when the supernet is widened (larger m) or given more shells (larger M).*

As with μ P, the corollary fixes the scales of the initialization and the updates, not the optimal HP values; Figure 3 verifies that the proxy’s tuned HP values remain near-optimal at both levels.

μ -ELASTIC for other elastic architectures. μ -ELASTIC carries over to other elastic architectures, but m and s are defined per architecture. MatFormer (Devvrit et al., 2024) keeps the model width fixed and nests only the FFN hidden width, so only its FFN down projection has a fan-in that varies with the sub-network. It uses $m = D_{\text{model}}/d_0$, with s the sub-network’s FFN hidden width in units of a base FFN width; the down projection takes the factor ms , and the other hidden matrices and the readout take m (Table 6). Fully elastic models, a transformer adaptation of slimmable networks (Yu et al., 2018), slice shared dense matrices along the residual, attention and FFN widths, so every hidden matrix has a varying fan-in; their m and s follow Equation 7, and every hidden matrix takes the factor ms (Table 5). StairFormer (Trost et al., 2026), with equal-width groups, takes Table 1 with $\mathcal{S} = \{1, \dots, M\}$ in its hidden matrices. Theorem 1 and Corollary 1 extend to all three (Appendix C.5), and Figure 8 compares MatFormer, fully elastic models and SHELLELASTIC.

3.3 TRAINING AND INFERENCE

Recipe. (1) Tune $(\sigma_0, \eta_0, \lambda_0)$ on a non-elastic proxy of width d_0 . (2) Choose the base width g and the supported set $\mathcal{S}$. (3) Compute $m = g/d_0$ and instantiate the parameterization in Table 1. (4) Train the supernet on the joint objective below.

Joint objective. At step τ , with selected widths $\mathcal{A}_\tau \subseteq \mathcal{S}$ and loss weights $\omega_s \geq 0$,

$$\mathcal{L}_\tau = \frac{\sum_{s \in \mathcal{A}_\tau} \omega_s \ell_s}{\sum_{s \in \mathcal{A}_\tau} \omega_s}, \quad \ell_s = \text{CE}(z_s, y) + \lambda_z \mathbb{E} \left[\left(\log \sum_v e^{z_{s,v}} \right)^2 \right], \quad (8)$$

where z_s are the logits of Equation 6, λ_z is the z -loss coefficient, the expectation runs over the non-padded target positions, and every ℓ_s backpropagates into the shared trunk. The narrower widths may be selected by any rule, but every $\mathcal{A}_\tau$ contains the full width M , so the trunk always runs at width D_{model} and every trunk parameter receives a gradient at every step.

Cost and scalability. Let $C_{\text{trunk}}(s)$ and $C_{\text{out}}(s)$ be the forward costs of the trunk and of one terminal norm and readout at width sg . Training the family jointly and training each sub-network separately cost

$$C_{\text{joint}} = C_{\text{trunk}}(M) + \sum_{s \in \mathcal{S}} C_{\text{out}}(s), \quad C_{\text{separate}} = \sum_{s \in \mathcal{S}} [C_{\text{trunk}}(s) + C_{\text{out}}(s)]. \quad (9)$$

By Proposition 1, one trunk term serves every width, whereas C_{separate} pays one per width, so the gap grows with $|\mathcal{S}|$ and C_{joint} stays comparable to training the widest sub-network alone. The same holds for memory: all readout gradients combine in one trunk backward pass, so the trunk’s activations, parameters and optimizer state are those of the widest one. Training all sub-networks at every step works best (Appendix J) and gives every step the same computation graph, parameter access pattern and workload. In MatFormer and fully elastic models, hidden states are not nested, so each width needs its own trunk pass: sampling widths per step (Devvrit et al., 2024; Yu & Huang, 2019) varies the workload, while training all widths at every step (Yu et al., 2018) pays C_{separate} .

Inference. For a chosen width $s \in \mathcal{S}$, we retain the embedding prefix, the core and shells up to s , and the matching terminal norm and readout, keeping the forward multipliers of Table 1. By Proposition 1, the extracted model reproduces its supernet logits exactly and runs as a standalone transformer of width sg without further training.

4 EXPERIMENTS

After describing the setup (Section 4.1), we evaluate the sub-networks of one **SHELLELASTIC** run against parameter-matched MatFormer, fully elastic and separately trained models, and μ -**ELASTIC** against SP on every elastic family (Section 4.2). We then compare **SHELLELASTIC** with StairFormer and run an architecture search at equal parameters and width count (Section 4.3); finally, we ablate the training and design choices to derive practical guidance for elastic pretraining (Section 4.4).

4.1 EXPERIMENTAL DETAILS

Models. Our primary model is a **SHELLELASTIC** supernet with base width $g = 128$ and supported set $\mathcal{S} = \{4, 5, 6, 7, 8\}$: a 512-dimensional core followed by four 128-dimensional shells, giving five sub-networks of widths 512 to 1024. We report main results at three scales, with sub-networks of about 119M–285M, 375M–906M and 675M–1.69B parameters (**SHELLELASTIC** widths 512–1024, 1024–2048 and 1024–2048, respectively). All models are 32-layer transformers with gated attention and SwiGLU FFNs. Tables 7 and 8 in Appendix E list all configurations.

Datasets. All models are trained on DCLM-Baseline (Li et al., 2024), a filtered Common Crawl corpus, with the GPT-NeoX-20B tokenizer (Black et al., 2022) and a sequence length of 2,048. By default, a run takes 25,000 steps of 128 sequences (6.55B tokens). This budget gives every sub-network more than 22 tokens per parameter, above the compute-optimal ratio of about 20 (Hoffmann et al., 2022). We monitor the training and validation losses (per-token cross-entropy in nats) of every sub-network, the latter on held-out C4 text (Raffel et al., 2020; Dodge et al., 2021). Following OLMoE (Muennighoff et al., 2025) and DCLM, we also probe 5-shot accuracy on ARC-Easy (Clark

Table 2: Final validation loss and mean 5-shot accuracy over 13 downstream tasks at three scales, labeled by sub-network size. Entries at the 119M–285M and 375M–906M scales average three runs; those at 675M–1.69B are single runs. [†]Five separately trained models, together taking $5\times$ the steps of any other row. Bold marks the best of the four methods that train all five sizes in one run.

		Val. loss ↓						5-shot acc. (%) ↑					
Method		Low	Medium	High	Extra	Max	Avg.	Low	Medium	High	Extra	Max	Avg.
119M–285M	Standalone [†]	3.369	3.308	3.264	3.219	3.191	3.270	35.6	35.8	36.6	37.1	37.1	36.5
	Sampled standalone	3.667	3.609	3.583	3.532	3.524	3.583	33.8	33.4	34.0	34.2	34.4	34.0
	MatFormer	3.425	3.357	3.325	3.312	3.314	3.347	35.1	35.2	35.8	35.9	35.9	35.6
	Fully elastic	3.429	3.354	3.315	3.295	3.290	3.336	34.9	35.5	35.7	35.9	36.0	35.6
	SHELLELASTIC	3.409	3.338	3.294	3.264	3.240	3.309	35.5	36.6	36.6	36.9	36.9	36.5
375M–906M	Standalone [†]	3.007	2.963	2.926	2.897	2.870	2.933	40.4	41.8	42.5	43.2	43.1	42.2
	Sampled standalone	3.208	3.175	3.141	3.115	3.093	3.147	37.7	37.8	38.0	38.7	39.0	38.2
	MatFormer	3.072	3.005	2.975	2.963	2.963	2.996	39.7	40.3	40.4	41.2	41.3	40.6
	Fully elastic	3.052	2.995	2.966	2.952	2.948	2.983	40.3	41.0	41.6	42.0	42.0	41.4
	SHELLELASTIC	3.055	3.000	2.967	2.943	2.925	2.978	40.8	41.4	42.0	42.6	43.0	41.9
675M–1.69B	Standalone [†]	2.912	2.859	2.821	2.793	2.767	2.830	43.4	43.9	44.7	46.2	46.1	44.8
	Sampled standalone	3.037	2.998	2.970	2.956	2.924	2.977	40.2	40.8	41.6	41.0	43.2	41.4
	MatFormer	2.914	2.872	2.849	2.839	2.838	2.862	42.8	43.3	43.4	44.0	43.6	43.4
	Fully elastic	2.940	2.884	2.854	2.838	2.833	2.870	42.1	43.4	43.9	44.2	44.6	43.6
	SHELLELASTIC	2.914	2.866	2.835	2.813	2.796	2.845	43.7	43.7	43.9	44.3	44.5	44.0

et al., 2018), PIQA (Bisk et al., 2020) and HellaSwag (Zellers et al., 2019) during training and evaluate the final sub-networks on 13 datasets spanning world knowledge, symbolic problem solving, commonsense reasoning, language understanding and reading comprehension (Appendix F).

Baselines. We compare **SHELLELASTIC** against five baselines, each providing networks of five sizes: Low, Medium, High, Extra and Max. (i) *MatFormer* (Devvrit et al., 2024) nests only the FFN hidden width. (ii) *Fully elastic*, a transformer adaptation of slimmable networks (Yu et al., 2018), nests the residual, attention and FFN widths by slicing shared dense matrices. (iii) *StairFormer* (Trost et al., 2026) shares the prefix structure of **SHELLELASTIC** but uses equal-width groups and normalizes over cumulative prefixes. (iv) *Standalone* trains five separate models, each on the full budget, and thus takes $5\times$ the steps of one elastic run. (v) *Sampled standalone*, the standalone baseline of MatFormer, trains the five *Standalone* models within the steps of one elastic run. MatFormer, the fully elastic model and sampled standalone sample one sub-network uniformly at random per step, whereas StairFormer and **SHELLELASTIC** train all sub-networks in one full-width pass at every step. **In the main results, all baselines match the **SHELLELASTIC** sub-networks in parameter count to within 3% (Table 10).** Appendix E details the baselines’ architecture and training, and Appendix D gives their parameterization.

Hyperparameters. All models are trained with AdamW. Unless otherwise specified, they use the base learning rate $\eta_0 = 2 \times 10^{-3}$, initialization scale $\sigma_0 = 3.3 \times 10^{-3}$ and weight decay $\lambda_0 = 0.01$, with μ -ELASTIC for the elastic models and μ P for the non-elastic ones, both with $d_0 = 128$. **SHELLELASTIC** has $m = 1$ and trains all five widths at every step with loss weights (1, 1, 1, 1, 4) (Equation 8). Main results at the 119M–285M and 375M–906M scales are averaged over three runs with the same seed; compute limits the 675M–1.69B scale to one run. Ablations vary only the setting under study; Table 9 in Appendix E lists the remaining HPs and the hardware.

4.2 MAIN RESULTS

Comparison with baselines across scales. Table 2 compares **SHELLELASTIC** with MatFormer, the fully elastic model and the two standalone baselines at three scales (curves and per-task results in Appendix G); all three elastic models use μ -ELASTIC (Appendix D). **SHELLELASTIC** is the supernet of Section 4.1 at 119M–285M, doubles its base width to $g = 256$ at 375M–906M, and also raises the FFN expansion from $e = 1$ to 4 at 675M–1.69B; these two scales train for 50,000 and 100,000 steps of 256 sequences (Appendix E). Among the methods that train all five sizes in one run, **SHELLELASTIC** has the lowest mean validation loss (0.028, 0.005 and 0.018 nats below the best baseline) and the highest mean 5-shot accuracy at every scale, and is best or tied-best at

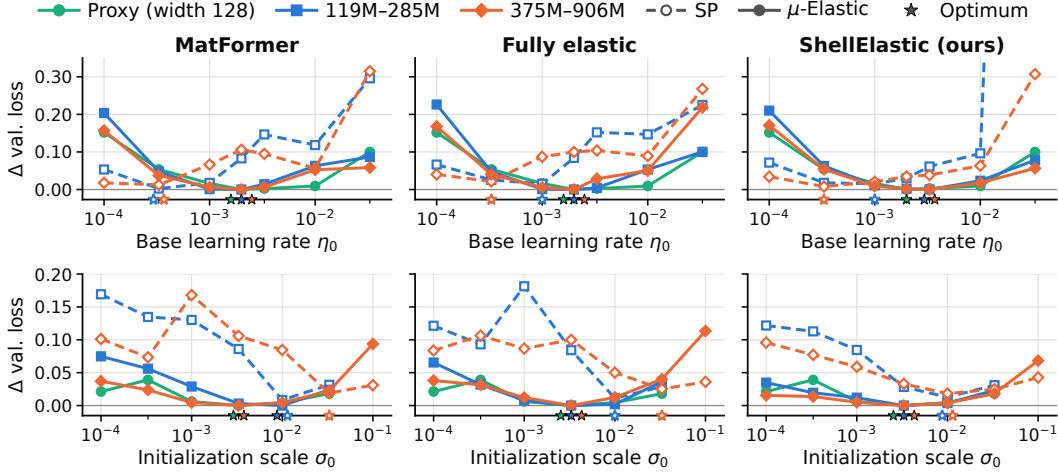


Figure 3: Sweeps of the learning rate η_0 and initialization scale σ_0 , the other at its base value, for the proxy and each elastic model at both scales under μ -ELASTIC and SP. Δ val. loss is the mean validation loss of the five sub-networks minus the minimum of the μ -ELASTIC line of the same model and scale (for the proxy, its own line). Appendix H shows the sweeps for each sub-network.

Table 3: Effect of μ -ELASTIC on each elastic model at the 119M–285M and 375M–906M scales. For each model, the first row is the best of the 12 (13 at 375M–906M) SP configurations of Figure 3 at that scale (lowest mean validation loss), and the w/ μ -ELASTIC row is its result in Table 2, with the base values of Section 4.1 and no re-tuning. Table 28 gives the values with four decimals.

		Train loss ↓						Val. loss ↓					
Method		Low	Medium	High	Extra	Max	Avg.	Low	Medium	High	Extra	Max	Avg.
119M–285M	MatFormer	3.261	3.184	3.157	3.140	3.139	3.176	3.434	3.360	3.327	3.314	3.311	3.349
	w/ μ -ELASTIC	3.254	3.182	3.156	3.141	3.144	3.175	3.425	3.357	3.325	3.312	3.314	3.347
	Fully elastic	3.272	3.192	3.154	3.130	3.126	3.175	3.445	3.368	3.325	3.303	3.297	3.348
	w/ μ -ELASTIC	3.257	3.178	3.145	3.123	3.120	3.165	3.429	3.354	3.315	3.295	3.290	3.336
	SHELLELASTIC	3.247	3.180	3.136	3.104	3.079	3.149	3.423	3.354	3.308	3.275	3.249	3.322
	w/ μ -ELASTIC	3.230	3.164	3.123	3.094	3.071	3.137	3.409	3.338	3.294	3.264	3.240	3.309
375M–906M	MatFormer	2.914	2.841	2.815	2.796	2.793	2.832	3.089	3.019	2.988	2.974	2.970	3.008
	w/ μ -ELASTIC	2.899	2.828	2.803	2.786	2.787	2.821	3.072	3.005	2.975	2.963	2.963	2.996
	Fully elastic	2.906	2.842	2.813	2.790	2.786	2.827	3.081	3.020	2.987	2.969	2.963	3.004
	w/ μ -ELASTIC	2.879	2.817	2.793	2.773	2.771	2.807	3.052	2.995	2.966	2.952	2.948	2.983
	SHELLELASTIC	2.888	2.830	2.796	2.770	2.748	2.806	3.070	3.009	2.973	2.946	2.924	2.984
	w/ μ -ELASTIC	2.875	2.822	2.790	2.767	2.749	2.801	3.055	3.000	2.967	2.943	2.925	2.978

12 of the 15 tiers in validation loss and 14 in 5-shot accuracy. Its lead grows with size, from at most 0.016 nats at Low to 0.023–0.050 at Max, likely because the Max-only parameters of the elastic baselines receive a gradient on only one step in five on average, whereas every parameter of SHELLELASTIC receives one at every step. At a fixed step budget, weight sharing also helps: the fully elastic model beats sampled standalone, its unshared counterpart, by 0.11 to 0.25 nats on average. With a fifth of their optimizer steps, SHELLELASTIC comes within 0.039, 0.045 and 0.014 nats of the standalone models in mean validation loss and within 0.8 points in mean 5-shot accuracy. Appendix E.4 analyzes the training and inference cost of all methods at 119M–285M.

Transferability of μ -ELASTIC. Under μ -ELASTIC, the proxy’s optimum, $\eta_0 = 2 \times 10^{-3}$ and $\sigma_0 = 3.3 \times 10^{-3}$ (the base values of Section 4.1), transfers to all three elastic models at both levels of μ -Transfer (Figure 3, Corollary 1). At the 119M–285M scale, both values are optimal or adjacent to the optimum (η_0 : exact for MatFormer and the fully elastic model, adjacent for SHELLELASTIC; σ_0 : exact for the fully elastic model and SHELLELASTIC, adjacent for MatFormer). At the 375M–906M scale, which enlarges m , M or both and doubles the batch size and steps (Appendix E), the same holds, and σ_0 becomes exact for MatFormer too. The base values are thus within 0.003 nats of the best point of every μ -ELASTIC sweep and within 0.005 nats for every sub-network (Appendix H). Under SP, the optimum departs from the proxy’s and shifts with scale, as Section 3.2 anticipates,

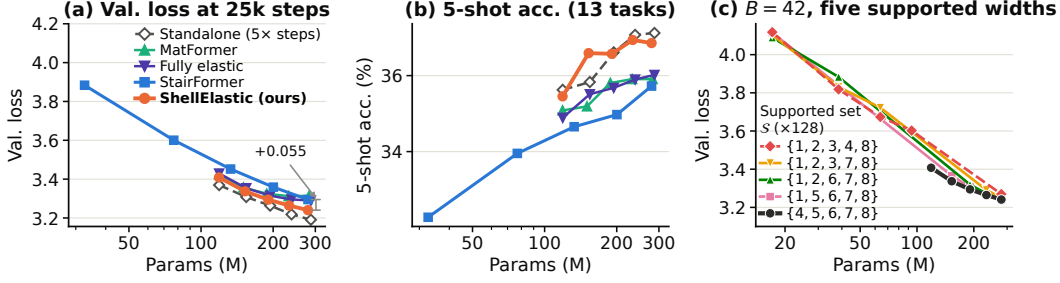


Figure 4: Architecture comparison and search against parameters at step 25,000. (a, b) Comparison with StairFormer: validation loss and average 5-shot accuracy (%) over 13 tasks. (c) Architecture search: validation loss of each supported width. Full results are in Appendices I.1 and I.2.

and at the base values SP is 0.03–0.11 nats worse. Weight decay transfers as well (Figure 15 in Appendix H): under μ -ELASTIC, the mean loss of SHELLELASTIC is flat over the same range as the proxy’s, from $\lambda_0 = 10^{-3}$ to 10^{-1} (within 0.003 nats), whereas SP is higher at every swept value.

Simultaneous maximal updates of μ -ELASTIC. Without width-dependent scaling, re-tuning one global value only trades one width against another (Proposition 2). Table 3 tests this limitation at the 119M–285M and 375M–906M scales, giving SP the best of its 12 (13 at 375M–906M) configurations in Figure 3 and μ -ELASTIC only the base values. At the 119M–285M scale, for the fully elastic model and SHELLELASTIC, μ -ELASTIC still gives lower training and validation losses at every tier, by 0.011 and 0.013 nats in mean validation loss, as it keeps all widths maximally updated at once (Theorem 1). At the 375M–906M scale, tuned SP remains 0.006–0.021 nats worse than μ -ELASTIC for all three models (Appendix H). Under tuned SP, SHELLELASTIC still beats both baselines by 0.026–0.027 and 0.020–0.024 nats at the two scales, so its lead is not due to μ -ELASTIC alone.

4.3 ARCHITECTURE COMPARISON AND SEARCH

Comparison with StairFormer. StairFormer (Trost et al., 2026) nests sub-networks as prefixes of one forward pass; it uses equal-width groups, cumulative prefix normalization and one shared readout. Here it has $g = 208$ and $\mathcal{S} = \{1, 2, 3, 4, 5\}$ in our notation, whereas SHELLELASTIC has a 512-wide core and four 128-wide shells ($\mathcal{S} = \{4, 5, 6, 7, 8\}$), group-local normalization and per-width readouts. Like SHELLELASTIC, it trains all sub-networks at every step; we give it the same μ -ELASTIC and HPs (Section 4.1). Its narrower sub-networks are much smaller (32.6M to 200.4M); only its Max matches SHELLELASTIC in size. At the parameter count of each SHELLELASTIC sub-network, its validation loss is 0.055 to 0.083 nats higher (Figure 4a; curves in Figure 17), and even its Max, with $1.5\times$ the parameters, is 0.001 nats above the High of SHELLELASTIC. At Max, SHELLELASTIC also leads in average 5-shot accuracy, 36.9 against 35.7 (Figure 4b and Table 29).

Architecture search at equal parameters and width count. Since $\mathcal{S}$ fixes both the extractable widths and the parameter count (a hidden matrix keeps $B(\mathcal{S}) = \sum_k (s_k - s_{k-1})s_k$ of its M^2 blocks of size $g \times g$), we ask where extra width should go when B and the number of supported widths are fixed. At $B = 42$ with five supported widths, each candidate set has four groups of width g and one of width $4g$ and differs only in where this wide group sits (block masks in Figure 18); at $B = 39$ with six, the wide group has width $3g$ (Appendix I.2). We train every supported width at every step and compare the sets at each width they share with equal parameter counts (Figure 4c). The core is the best place for the wide group: the core placement has the lowest loss, or is within 0.006 nats of it, at every shared width, whereas a shell placement makes the first width containing the wide group 0.025 to 0.052 nats worse. This motivates the wide core and thin shells of SHELLELASTIC.

4.4 ABLATION STUDIES

All ablations use the 119M–285M supernet of Section 4.1 and change one setting at a time; the default results are those of Table 2. Of the settings we ablate, sub-network selection, in-place distillation, ensembles of the sub-networks, stage-residual gradient scaling and the trunk gradient of the narrower widths can change the mean validation loss by 0.010 to 0.20 nats, whereas the loss weights, normalization, readouts, base width and head dimension, and optimization HPs change it by at most 0.007 nats. We discuss the first four below (Table 4) and the others in Appendix J.

Table 4: Ablations at the 119M–285M scale: final validation loss and mean 5-shot accuracy over 13 downstream tasks and the five tiers. Each block ends with the default, and bold marks the best value of each column within a block. Appendix J gives all ablations.

Setting	Val. loss ↓						5-shot acc. ↑
	Low	Medium	High	Extra	Max	Avg.	Avg.
<i>Selected sub-networks per step $\mathcal{A}_\tau$</i>							
Sandwich: Low, Max and one other	3.4019	3.3617	3.3183	3.2816	3.2395	3.3206	36.1
Random: one of the five sub-networks	3.4311	3.3786	3.3542	3.3410	3.3356	3.3681	34.8
Max + random: Max and one other	3.4944	3.4079	3.3447	3.2956	3.2397	3.3564	35.5
All: all five sub-networks (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>In-place distillation from Max</i>							
Forward KL, weight 0.1	3.4047	3.3355	3.2937	3.2641	3.2410	3.3078	36.4
Forward KL, weight 0.5	3.3920	3.3287	3.2931	3.2675	3.2482	3.3059	36.6
Forward KL, weight 1	3.3748	3.3184	3.2873	3.2653	3.2483	3.2988	36.1
Reverse KL, weight 0.5	3.3868	3.3261	3.2887	3.2632	3.2437	3.3017	35.9
Symmetric KL, weight 0.5	3.3875	3.3256	3.2901	3.2629	3.2435	3.3019	36.0
None (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>Ensembles of the five sub-networks</i>							
Learned per-token width router	3.7735	3.5611	3.4137	3.3055	3.2269	3.4562	35.1
Distillation from loss-weighted ensemble	3.3902	3.3275	3.2926	3.2673	3.2471	3.3049	37.0
Distillation from soft oracle	3.3982	3.3350	3.2980	3.2709	3.2501	3.3105	36.3
None (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>Stage-residual gradient scaling</i>							
$\gamma = 0$	3.4053	3.3740	3.3530	3.3348	3.3115	3.3557	36.1
$\gamma = 0.3$	3.3806	3.3317	3.2991	3.2762	3.2485	3.3072	36.6
$\gamma = 0.5$	3.3939	3.3360	3.2967	3.2700	3.2447	3.3083	36.0
$\gamma = 0.8$	3.4030	3.3365	3.2946	3.2658	3.2419	3.3084	37.0
$\gamma = 1$ (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5

Selected sub-networks per step. Besides training all five sub-networks at every step (All, the default), we test three rules for $\mathcal{A}_\tau$ in Equation 8: *Sandwich* (Low and Max plus one other), *Max + random* (Max plus one other) and *Random* (one sub-network, as in MatFormer), drawing the random ones uniformly at random. The loss of each sub-network follows how often it is trained: Sandwich and Max + random match All at Max but are up to 0.024 and 0.086 nats worse at those they select less often, and Random is worse at every sub-network, by 0.095 nats at Max. All gives the lowest mean loss, 0.012 to 0.059 nats below the others, and the highest mean 5-shot accuracy.

In-place distillation. Elastic recipes often distill the narrower sub-networks from the widest one (Yu & Huang, 2019). We add to the losses of Low to Extra a KL divergence toward the output distribution of Max (forward KL with weights 0.1 to 1; reverse and symmetric KL with weight 0.5). This lowers the loss of Low by up to 0.034 nats and the mean loss by up to 0.010 nats, the largest gain of any ablation, but raises that of Max by up to 0.008 nats, improves the mean 5-shot accuracy by at most 0.1 points and makes a step 2.2 to 2.6 times as long, so we do not use it.

Ensembles of the sub-networks. One trunk pass yields the output distributions p_s of all five sub-networks, which may err on different tokens, so a per-token ensemble $\sum_s \pi_s p_s$ could beat Max as a predictor at little extra cost, or replace it as a stronger teacher (details in Appendix J).

- *Ensemble as predictors.* We evaluate four ensembles of a trained supernet: uniform weights, confidence weights ($\pi_s \propto \max_v p_s(v)$), the most confident readout per token, and the best fixed ensemble, whose weights minimize the loss. We also train a router, an MLP that maps the last-layer hidden state of Max to per-token weights π_s , together with its own supernet on the π -weighted sum of the five losses. None of the ensemble methods beat Max alone (Figure 5, Table 33): the best fixed ensemble only matches it, the other three are 0.03 to 0.065 nats worse in train loss, and the router puts all weight on Max within the first 50 steps (Figure 5b), so Max improves by 0.013 nats while the other sub-networks, now barely trained, lose up to 0.365 nats (Figure 5c).
- *Oracle.* An oracle that picks per token the readout with the highest target probability is 0.19 nats below Max (Table 34), as a narrower sub-network is better on 71% of tokens (Figure 5e). Two

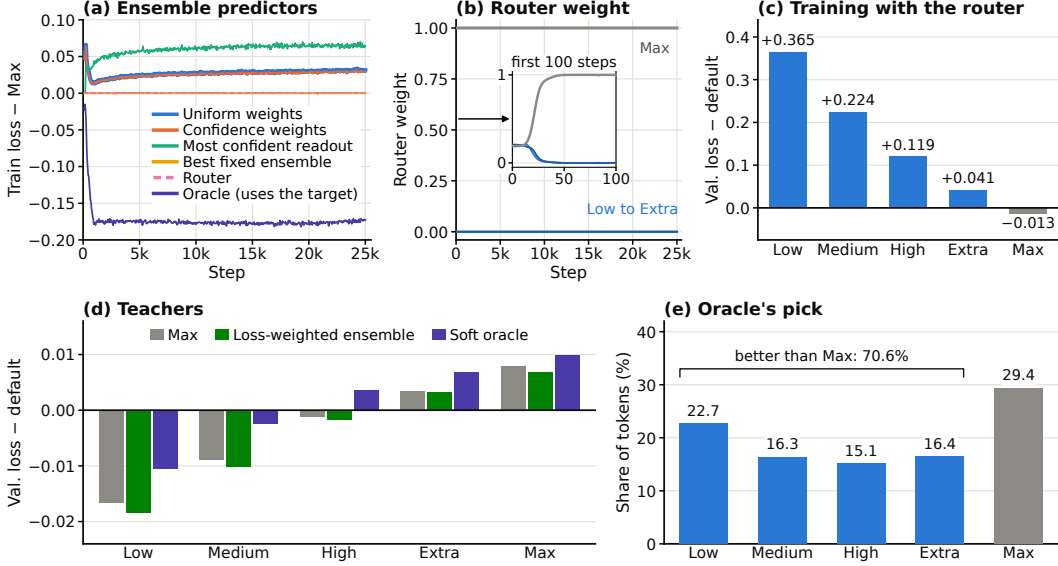


Figure 5: Ensembles of the sub-networks. (a) Train loss minus that of Max during training for the five predictors and the oracle, which uses the target. (b) Router weight on each tier. (c) Validation loss of each tier minus the default when training with the router. (d) Validation loss of each tier minus the default when distilling toward Max, the loss-weighted ensemble or the soft oracle. (e) Share of held-out tokens on which each tier assigns the target the highest probability. The default is the mean of the three **SHELL**ELASTIC runs of Table 2.

supernets trained separately with the same settings differ far more: averaging their Max lowers the loss by 0.056 nats at twice the cost, and their oracle gains 0.23 nats against 0.19 for the five sub-networks on held-out text. Our ensembles show that finding the better sub-network per token without the target is hard, but we believe a better ensemble could close part of this gap, a promising way to substantially improve **SHELL**ELASTIC at almost no extra cost.

- *Ensemble as teachers.* Every sub-network, including Max, distills (forward KL, weight 0.5) toward the loss-weighted ensemble ($\pi_s \propto \omega_s$), which predicts 0.012 nats worse than Max, or the soft oracle ($\pi_s \propto p_s(y)^\beta$), a smoothed oracle that predicts 0.14 nats better (Table 33). Yet neither teacher helps Max, not even the soft oracle: like in-place distillation with the same KL (Figure 5d), they improve Low by up to 0.018 nats and worsen Max by up to 0.010 nats.

Stage-residual gradient scaling. Since the readout is linear (Equation 6), the logits of width s_k are a sum of k terms, one from the last-layer coordinates of each of its groups, so the supernet can be viewed as a sequence of stages: the core gives a base prediction and each shell adds a correction, as in residual learning (He et al., 2016). To make each shell learn this correction, we keep the forward pass but, in the backward pass, scale by γ the gradient that each sub-network’s loss sends to the trunk through the narrower sub-network it contains; the gradient through its own shell is unchanged (details in Appendix J). Thus $\gamma = 1$ is the usual joint training of Equation 8, while with $\gamma = 0$ each loss reaches the trunk only through its own shell, which therefore learns only to correct the prediction of the narrower sub-network. A γ of 0.3 to 0.8 lowers the loss of Low by up to 0.028 nats but raises those of Extra and Max by up to 0.012 nats, leaving the mean loss within 0.002 nats, whereas $\gamma = 0$ raises the mean loss by 0.047 nats, as the last-layer features of the narrower groups no longer adapt to the wider losses; we keep $\gamma = 1$. The stage view still points to a property that MatFormer and fully elastic models lack: since no group reads a later one and every group is normalized on its own, a new shell can be appended to a trained supernet and trained as a new stage with all trained weights frozen, so every trained sub-network keeps its exact outputs.

5 CONCLUSION

In **SHELL**ELASTIC, one full-width pass gives the exact hidden states of every sub-network, so joint training costs about as much as the widest one alone. With μ -ELASTIC and proxy-tuned HPs, one run yields maximally updated sub-networks that beat MatFormer and fully elastic models at matched

sizes and approach standalone models that together take five times the steps. Our experiments favor training all sub-networks at every step and a wide core, and suggest two directions: better sub-network ensembles, whose oracle is 0.19 nats below Max, and appending shells to a trained supernet.

AI USE STATEMENT

In this work, we used generative AI tools to provide critical ingredients for proving our mathematical claims, to assist in writing the proofs (Appendices B and C), and to implement parts of our method. Additionally, we used generative AI tools to draft parts of the paper, edit it for readability, generate and check the code for the result figures and tables, and cross-check the statements of the paper against the training code. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies training methods for language models on publicly available web text and does not involve human subjects. Elastic models reduce the cost of serving several model sizes, which lowers the energy cost of deployment; they inherit the biases of their training data like any language model.

REPRODUCIBILITY STATEMENT

Section 4.1 and Appendices E and F list every architectural, optimization and evaluation setting; Table 1 and Appendix D give the complete parameterization rules; Appendices B and C contain the proofs, and Appendix B.3 verifies Proposition 1 numerically; Appendices G, H, I and J give the complete main, transfer, architecture-search and ablation results. The training code, configurations and run logs used for all tables will be released.

REFERENCES

- Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.
- Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, pp. 95–136, 2022.
- Blake Bordelon, Lorenzo Noci, Mufan Li, Boris Hanin, and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit. In *International Conference on Learning Representations*, volume 2024, pp. 22088–22127, 2024.
- Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HylxE1HKwS>.
- Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5298–5311. PMLR, 21–27 Jul 2024. URL <https://proceedings.mlr.press/v235/cai24e.html>.
- Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.
- Fnu Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit S Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham M Kakade, Ali Farhadi, et al. Matformer: Nested transformer for elastic inference. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- Nolan Dey, Bin Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlivan, Boris Hanin, and Joel Hestness. Don’t be lazy: Completep enables compute-efficient deep transformers. *Advances in Neural Information Processing Systems*, 38:137707–137739, 2026.
- Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 1286–1305, 2021.
- Katie E Everett, Lechao Xiao, Mitchell Wortsman, Alexander A Alemi, Roman Novak, Peter J Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12666–12700. PMLR, 21–27 Jul 2024. URL <https://proceedings.mlr.press/v235/everett24a.html>.
- Gemma Team. Gemma 3n. <https://ai.google.dev/gemma/docs/gemma-3n>, 2025.
- Andrew Gordon, Zornitsa Kozareva, and Melissa Roemmele. Semeval-2012 task 7: Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pp. 394–398, 2012.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.
- Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS ’22*, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.
- Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33: 9782–9793, 2020.

- Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.
- Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.
- Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=Bkg6RiCqY7>.
- Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.
- Bruno Mlodozieniec, Pierre Ablin, Louis Béthune, Dan Busbridge, Michal Klein, Jason Ramapuram, et al. Completed hyperparameter transfer across modules, width, depth, batch and duration. In *International Conference on Learning Representations*, volume 2026, pp. 45300–45325, 2026.
- Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. Olmoe: Open mixture-of-experts language models. In *International Conference on Learning Representations*, volume 2025, pp. 62061–62121, 2025.
- Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.
- Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 2080–2094, 2021.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.
- Noam Shazeer. Glue variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.
- Jascha Sohl-Dickstein, Roman Novak, Samuel S Schoenholz, and Jaehoon Lee. On the infinite width limit of neural networks with a standard parameterization. *arXiv preprint arXiv:2001.07301*, 2020.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.
- Ali Taghibakhshi, Sharath Turuvekere Sreenivas, Saurav Muralidharan, Ruisi Cai, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Yoshi Suhara, Oluwatobi Olabiyi, Daniel Korzekwa, Mostofa Patwary, et al. Nemotron elastic: Towards efficient many-in-one reasoning llms. *arXiv preprint arXiv:2511.16664*, 2025.

- Avi Trost, Alexander Yun, John Cooper, Gabriel Orlanski, and Frederic Sala. Fully nested transformers. In *AdaptFM: Resource-Adaptive Foundation Model Inference*, 2026.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34:17084–17097, 2021.
- Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pp. 11727–11737. PMLR, 2021.
- Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs vi: Feature learning in infinite depth neural networks. In *International Conference on Learning Representations*, volume 2024, pp. 55099–55150, 2024.
- Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1803–1811, 2019.
- Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.
- Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.
- Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.
- Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the association for computational linguistics: NAACL 2024*, pp. 2299–2314, 2024.

APPENDIX CONTENTS

A	Architecture details	18
B	Exact computation reuse	21
B.1	Prefix-preserving maps	21
B.2	Proof of Proposition 1	21
B.3	Numerical verification and scope	22
C	Proofs of the μ-ELASTIC results	22
C.1	Assumptions and notation	22
C.2	Proof of Theorem 1	23
C.3	Proof of Proposition 2	24
C.4	Proof of Corollary 1	24
C.5	Extension to other elastic architectures	25
D	μ-ELASTIC for other elastic architectures	25
E	Training details	26
E.1	Model configurations	27
E.2	Optimization	27
E.3	Parameter counts	27
E.4	Training and inference cost	28
F	Evaluation details	30
G	Complete main results	32
H	Additional transfer results	44
I	Architecture comparison and search	48
I.1	Comparison with StairFormer	48
I.2	Architecture search at equal parameters and width count	48
J	Ablation studies	52

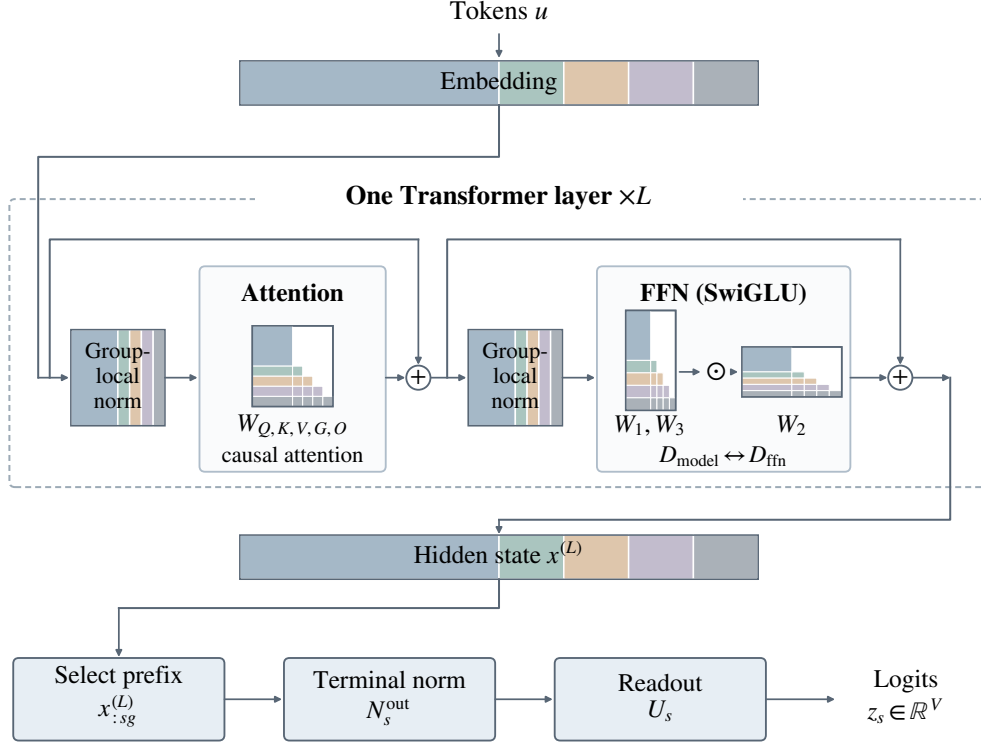


Figure 6: **The complete SHELLELASTIC model.** Embedding, one attention–FFN layer repeated L times, and the terminal norm and readout of the selected width. Group-local branch normalization and structured projections preserve every supported prefix through the trunk.

A ARCHITECTURE DETAILS

Figure 6 shows the complete **SHELLELASTIC** model, Figure 7 details its attention and FFN sub-layers, and Figure 8 places the three elastic families side by side. Weight matrices are drawn with output rows and input columns and colored only where a connection is allowed; white lines mark the supported widths, and black lines delimit the slices of a dense matrix that the narrower sub-networks use. All three families are drawn with $\mathcal{S} = \{4, 5, 6, 7, 8\}$, the default supported set of **SHELLELASTIC**, so that their matrices are directly comparable; Table 7 lists the dimensions of the trained models. The forward multipliers of Table 1 are omitted throughout, and the per-head query/key normalization and rotary position transformations appear only in Figure 7.

Comparison with other elastic architectures. MatFormer keeps the model width, attention, normalization, embedding and readout at full width and nests only the FFN hidden width. A fully elastic model slices dense matrices along both axes and normalizes over the active width. In both cases, channel mixing is dense, so the hidden states of a narrow sub-network are not contained in those of a wide one, and each width needs its own trunk pass. **SHELLELASTIC** replaces the dense matrices with block-lower-triangular ones and the full-width normalization with group-local normalization, which is exactly what lets the supported prefixes share a single trunk pass (Proposition 1).

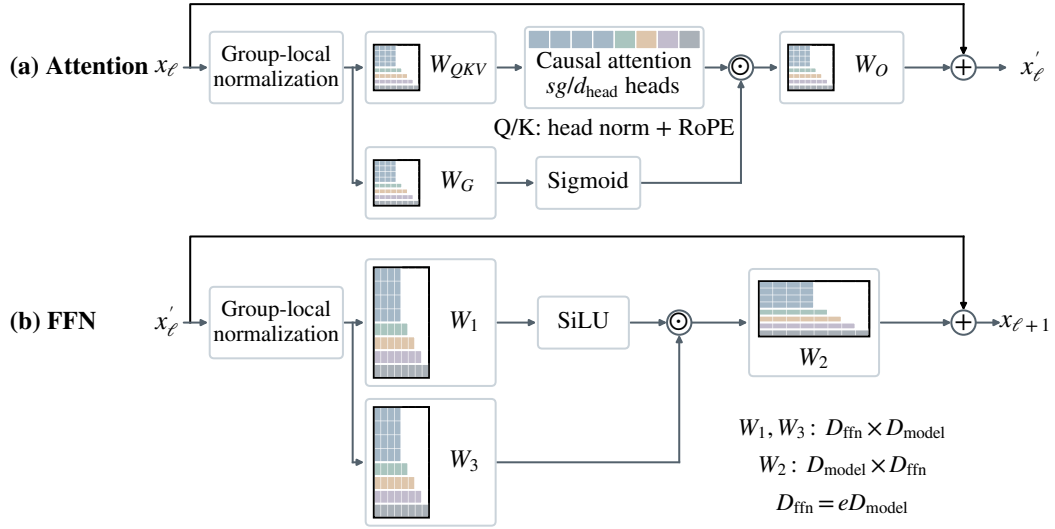


Figure 7: **Prefix-preserving attention and FFN.** Sub-network s uses the first sg/d_{head} attention heads. The FFN’s up, gate, and down projections expand and contract the same groups.

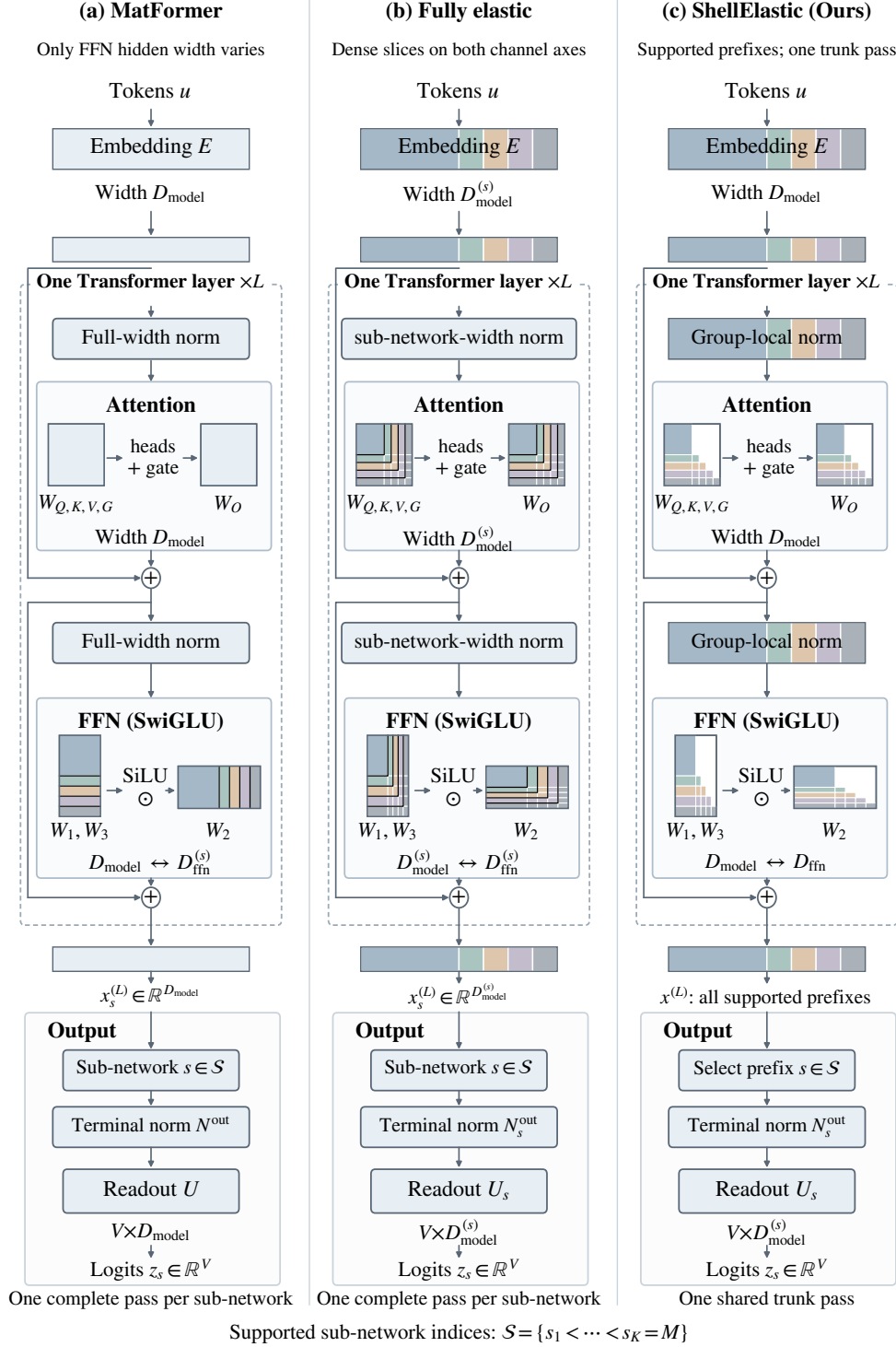


Figure 8: **Complete architecture comparison.** MatFormer varies the FFN hidden dimension; fully elastic models vary the model dimension and the FFN hidden dimension together by slicing dense matrices; **SHELLELASTIC** also varies both, but through block-lower-triangular projections in which each output group reads only a prefix of the input. Each layer frame repeats L times. Proportions are illustrative.

B EXACT COMPUTATION REUSE

This appendix proves Proposition 1. Appendix B.1 defines prefix-preserving maps and shows that the structured projections are prefix-preserving at the supported widths, Appendix B.2 composes the operations of the trunk, and Appendix B.3 checks the result numerically and delimits what it does and does not claim.

B.1 PREFIX-PRESERVING MAPS

Throughout this appendix, $s \leq t$ are supported widths in $\mathcal{S}$, and P_s keeps the first sg coordinates of every token’s hidden state (the first esg coordinates on the FFN hidden axis), as in Proposition 1. For a map F that the trunk applies at every supported width, F_s denotes its restriction to width sg , that is, the map that sub-network s applies.

Definition 1 (Prefix-preserving map). *A map F is prefix-preserving if*

$$P_s \circ F_t = F_s \circ P_s \quad \text{for all } s, t \in \mathcal{S}, s \leq t. \quad (10)$$

We first show that the structured projections of Section 3.1 have this property. The lemma allows different base granularities on the two axes, since the output groups of the FFN up and gate projections are e times as wide as their input groups, and those of the down projection e times as narrow.

Lemma 1 (Structured linear maps). *Consider a linear map with base input and output granularities g_{in} and g_{out} , in which output group k has $(s_k - s_{k-1})g_{\text{out}}$ rows and reads the first $s_k g_{\text{in}}$ input coordinates. Let P_s^{in} and P_s^{out} keep the first sg_{in} inputs and the first sg_{out} outputs, and let W_s be the top-left $sg_{\text{out}} \times sg_{\text{in}}$ block of the weight matrix. Then, with the row-vector convention,*

$$P_s^{\text{out}}(xW_t^\top) = P_s^{\text{in}}(x)W_s^\top, \quad s, t \in \mathcal{S}, \quad s \leq t. \quad (11)$$

Proof. Let $s = s_j$. The first sg_{out} outputs are those of the groups $k \leq j$, and output coordinate i of group k is

$$(xW_t^\top)_i = \sum_{c=1}^{s_k g_{\text{in}}} (W_t)_{ic} x_c. \quad (12)$$

Since $k \leq j$ gives $s_k \leq s_j = s$, this sum reads only the first sg_{in} inputs, and all of its weights lie in the top-left block W_s . Hence it equals $(P_s^{\text{in}}(x)W_s^\top)_i$. $\square$

Unsupported widths. At an index $u \notin \mathcal{S}$, there is a group with $s_{k-1} < u < s_k$. A cut at u keeps part of that group but removes inputs it is allowed to read (Equation 12), so Equation 11 does not hold in general, and the architecture guarantees no reusable sub-network there. For $\mathcal{S} = \{4, 5, 6, 7, 8\}$, the indices 1, 2, 3 lie inside the dense core and are not supported.

B.2 PROOF OF PROPOSITION 1

Proof. We show that every operation of a layer is prefix-preserving (Steps 1–4) and then compose the operations over the L layers (Step 5).

Step 1: projections. The query, key, value, gate and output projections of attention and the up, gate and down projections of the FFN are structured linear maps. Lemma 1 therefore applies to each of them, with $g_{\text{in}} = g_{\text{out}} = g$ for the attention projections, $(g_{\text{in}}, g_{\text{out}}) = (g, eg)$ for the FFN up and gate projections, and (eg, g) for the FFN down projection.

Step 2: group-local normalization. By Equation 3, $N(x)_{\mathcal{G}_k} = N_k(x_{\mathcal{G}_k})$ uses only the coordinates and gains of $\mathcal{G}_k$, so the first sg outputs depend only on the first sg inputs, and $P_s \circ N_t = N_s \circ P_s$. The same holds for any finer partition and for a cumulative prefix norm, whose statistics for a coordinate use all coordinates up to the end of its group or of its block of g coordinates (Equation 25); for any of the first sg coordinates, that end is at most sg . A norm over the full active width does not preserve prefixes, because adding a group changes the statistics of the earlier coordinates.

Step 3: attention. Since g is a multiple of d_{head} , every head lies inside one group, and sub-network s uses the first sg/d_{head} heads. By Step 1, their queries, keys and values depend only on the first

sg inputs. Query-key normalization, the rotary transformation and the softmax act within a head, so the heads that sub-network t adds leave the outputs of these heads unchanged. The sigmoid gate and the output projection of Equation 4 are covered by Steps 1 and 4.

Step 4: elementwise operations, residuals and embedding. The SwiGLU product and the other elementwise operations act coordinatewise, so they commute with P_s . A residual branch $x \mapsto x + F(x)$ with a prefix-preserving F is prefix-preserving, since $P_s(x + F_t(x)) = P_s(x) + F_s(P_s(x))$. The embedding satisfies $P_s(E_t[u]) = E_s[u]$ by construction.

Step 5: composition over layers. If F and G are prefix-preserving, then so is $G \circ F$:

$$P_s \circ G_t \circ F_t = G_s \circ P_s \circ F_t = G_s \circ F_s \circ P_s. \quad (13)$$

Each layer is a composition of the operations of Steps 1–4 and is therefore prefix-preserving, and by induction over the L layers, as in Proposition 2.3 of StairFormer (Trost et al., 2026), so is the trunk: $P_s \circ T_t = T_s \circ P_s$. Applying this identity to the embedded sequence and using Step 4,

$$P_s(T_t(E_t[u])) = T_s(P_s(E_t[u])) = T_s(E_s[u]), \quad (14)$$

which is the claim. $\square$

Consequence for the joint objective. The terminal norm N_s^{out} and the readout U_s need not commute with P_s , since N_s^{out} normalizes over all sg selected coordinates. They are applied *after* the prefix has been selected (Equation 6), so each sub-network’s terminal map receives exactly the input it would receive in a separate evaluation. Hence each ℓ_s , and the gradient of the joint objective of Equation 8, is the same function of the shared parameters as in separate evaluations, up to floating-point summation order.

B.3 NUMERICAL VERIFICATION AND SCOPE

Numerical verification. We instantiated the full trained configuration on the meta device to check module selection and shapes, and ran a reduced two-layer model on CPU with the same architecture switches, comparing one shared trunk pass against five separate evaluations of the same nested weights. The logits and the weighted cross-entropy agreed exactly; the largest absolute difference between the two sets of parameter gradients was 1.68×10^{-8} , and every trainable parameter received a gradient. The check replaces the fused SwiGLU and cross-entropy kernels of the GPU path with reference expressions, so it supports the implementation of Proposition 1, not its throughput.

Scope of the statement. The proposition assumes the dense-FFN graph of Section 3.1, a fixed supported set $\mathcal{S}$ and prefix-preserving normalization. Beyond these assumptions:

- dropout needs matched masks for an equality test;
- data-dependent routing or width-specific trunk modules need a separate argument;
- Equation 2 fixes the logical dependencies and not a storage layout or a kernel: every implementation of that map has the prefix property.

C PROOFS OF THE μ -ELASTIC RESULTS

This appendix proves Theorem 1, Proposition 2 and Corollary 1. Appendix C.1 states the assumptions and fixes the notation, Appendices C.2–C.4 give the three proofs, and Appendix C.5 extends Theorem 1 and Corollary 1 to the other elastic architectures of Section 3.2.

C.1 ASSUMPTIONS AND NOTATION

Assumption 1 (Heuristic scaling). *We use the heuristic framework of Appendix J of the μ Transfer paper (Yang et al., 2021):*

- (A1) *Inputs. The input to every hidden matrix has $\Theta(1)$ coordinates with bounded second moments and is independent of that matrix at initialization.*

- (A2) Central limit. If $w_1, \dots, w_n$ are i.i.d. with mean zero and standard deviation b and independent of $x_1, \dots, x_n = \Theta(1)$, then $\sum_{c=1}^n w_c x_c = \Theta(\sqrt{n} b)$.
- (A3) Law of large numbers. After a gradient step, the update ΔW of a hidden matrix is aligned with the activations it multiplies, so $\sum_{c=1}^n \Delta W_{jc} x_c = \Theta(n \delta)$ when $|\Delta W_{jc}| = \Theta(\delta)$.
- (A4) Adam. With learning rate ρ and a non-degenerate second-moment estimate, every coordinate of an Adam update has magnitude $\Theta(\rho)$, independently of the gradient scale.

Notation. Consider a hidden matrix and the block row $W_k \in \mathbb{R}^{|\mathcal{G}_k| \times sg}$ of Equation 2 that produces output group $\mathcal{G}_k$, whose supported boundary is $s = s_k$. Inside a sub-network $t \in \mathcal{S}$ with $t \geq s$, let x be the input of the matrix, and let

$$y_j = \sum_{c=1}^{sg} (W_k)_{jc} x_c, \quad j \in \mathcal{G}_k, \quad (15)$$

be the pre-activation of coordinate j . After one optimizer step, the update ΔW_k moves it by $\Delta y_j = \sum_{c=1}^{sg} (\Delta W_k)_{jc} x_c$. Read widths are measured in base widths: the block row of $\mathcal{G}_k$ reads $s_k = s$ of them, and the readout of sub-network s reads s . The proxy width d_0 and the base values $(\sigma_0, \eta_0, \lambda_0)$ are fixed, and the constants hidden in $\Theta(\cdot)$ do not depend on g, s, t, M, m or e .

C.2 PROOF OF THEOREM 1

Proof. **Step 1: the fan-in does not depend on the sub-network.** The sum in Equation 15 has exactly sg terms in every sub-network $t \geq s$: the block row reads no coordinate beyond sg (Equation 2), and sub-network t contains all of them because $tg \geq sg$. One initialization scale and one learning rate, set by $q(s) = ms = sg/d_0$ (Equation 7), therefore serve every sub-network that contains the group.

Step 2: initialization. Table 1 draws the entries of W_k i.i.d. with standard deviation $\sigma_0/\sqrt{ms} = \sigma_0\sqrt{d_0/(sg)}$. By (A1) and (A2),

$$\mathbb{E}[y_j^2] = sg \cdot \frac{\sigma_0^2 d_0}{sg} \cdot \Theta(1) = \Theta(\sigma_0^2 d_0). \quad (16)$$

Step 3: one AdamW step. The learning rate of W_k is $\eta_0/(ms) = \eta_0 d_0/(sg)$, so by (A4) every coordinate of its update satisfies $|(\Delta W_k)_{jc}| = \Theta(\eta_0 d_0/(sg))$, and by (A3)

$$\Delta y_j = \sum_{c=1}^{sg} (\Delta W_k)_{jc} x_c = \Theta\left(sg \cdot \frac{\eta_0 d_0}{sg}\right) = \Theta(\eta_0 d_0). \quad (17)$$

The decoupled weight decay of AdamW, which is scaled by the learning rate, multiplies each coordinate by

$$1 - \frac{\eta_0}{ms} \cdot \lambda_0 ms = 1 - \eta_0 \lambda_0 \quad (18)$$

per step, the same factor for every decayed parameter group and every width.

Step 4: FFN down projection. In the FFN down projection, the block row of $\mathcal{G}_k$ reads esg coordinates, and its output carries the forward multiplier $1/e$ (Table 1). With initialization $\sigma_0\sqrt{e}/\sqrt{ms}$ and learning rate $\eta_0/(ms)$, the computations of Steps 2 and 3 become

$$\mathbb{E}[y_j^2] = e^{-2} \cdot esg \cdot \frac{e \sigma_0^2 d_0}{sg} \cdot \Theta(1) = \Theta(\sigma_0^2 d_0), \quad (19)$$

$$\Delta y_j = \Theta\left(e^{-1} \cdot esg \cdot \frac{\eta_0 d_0}{sg}\right) = \Theta(\eta_0 d_0), \quad (20)$$

so both scales are also free of e .

Step 5: readouts and embedding. The readout U_s reads the sg coordinates of $N_s^{\text{out}}(x_{:sg}^{(L)})$ (Equation 6), which is exactly the fan-in of the group with boundary s , and Table 1 gives it the same initialization and learning rate, following the per-layer scaling of Everett et al. (2024). Steps 2 and 3

therefore give the same two bounds for its logits at initialization and for their change under the update of U_s , and the learned scalar α_s rescales them only by $\Theta(1)$. The embedding is a lookup over a fixed vocabulary of size V , which does not change with width, and reading its first sg columns does not change the scale of any coordinate.

Step 6: joint training. Under the joint objective of Equation 8, the gradient of W_k sums the contributions of the selected sub-networks $t \geq s$. By Proposition 1, all of them feed W_k the same input x , so every contribution is an outer product with x , and the summed gradient, whatever its scale, keeps each row aligned with x ; (A3) therefore applies, and by (A4) the Adam update has coordinates of size $\Theta(\eta_0/(ms))$. The group receives a gradient whenever some selected $t \geq s$ has $\omega_t > 0$, which holds for $t = M$ when $\omega_M > 0$, and the readout U_s whenever $s \in \mathcal{A}_r$ and $\omega_s > 0$. Steps 2–5 therefore hold for the jointly trained supernet.

The constants $\Theta(\sigma_0^2 d_0)$ and $\Theta(\eta_0 d_0)$ of Equations 16 and 17 depend on neither s , t , M nor m , which proves (i) and (ii) for every hidden group and every readout. $\square$

C.3 PROOF OF PROPOSITION 2

Proof. Consider a hidden matrix whose input satisfies (A1), such as a projection that reads a group-local norm, and let all of its groups share the initialization standard deviation b , the learning rate ρ and the forward multiplier, which we absorb into b and ρ .

Fixed ratios. The group with boundary s sums over sg coordinates (Step 1 of Appendix C.2), so by (A2)–(A4)

$$y_j = \Theta(\sqrt{sg}b), \quad \Delta y_j = \Theta(sg\rho). \quad (21)$$

For boundaries $s < s'$, the pre-activations and updates of the two groups therefore differ by the factors $\Theta(\sqrt{s'/s})$ and $\Theta(s'/s)$, which do not depend on (b, ρ) : changing the shared pair rescales all groups together and leaves these factors unchanged.

Calibrations. Calibrating the shared pair for the group with boundary $r \in \mathcal{S}$ means taking the values of Table 1 for that group, $b = \sigma_0 \sqrt{d_0/(rg)}$ and $\rho = \eta_0 d_0/(rg)$. By Equation 21, the group with boundary s then has

$$y_j = \Theta(\sigma_0 \sqrt{d_0} \cdot \sqrt{s/r}), \quad \Delta y_j = \Theta(\eta_0 d_0 \cdot s/r), \quad (22)$$

so it deviates from the scales $\Theta(\sigma_0 \sqrt{d_0})$ and $\Theta(\eta_0 d_0)$ of Theorem 1 by the factors $\sqrt{s/r}$ and s/r . For $r = M$, the group with boundary s has pre-activations of size $\Theta(\sqrt{s/M})$ and updates of size $\Theta(\eta_0 s/M)$, up to the constants $\sigma_0 \sqrt{d_0}$ and d_0 . For $r = s_1$, it is inflated by $\sqrt{s/s_1}$ and s/s_1 , and the outermost group by $\sqrt{M/s_1}$ and M/s_1 .

No shared pair is uniformly maximal. Let the core and the outermost group deviate from the scales of Theorem 1 by the factors a_1 and a_K at initialization. By the fixed ratios, $a_K/a_1 = \Theta(\sqrt{M/s_1})$, so $\max(a_1, a_1^{-1}, a_K, a_K^{-1})$ grows at least as $(M/s_1)^{1/4}$; the same argument for the updates gives $(M/s_1)^{1/2}$. Over supernets with unbounded M/s_1 , no shared pair keeps both factors within constants independent of M , so none is maximal in the sense of Theorem 1 for every group. When M/s_1 is bounded, the mismatch is a bounded factor: for $\mathcal{S} = \{4, 5, 6, 7, 8\}$, $M/s_1 = 2$, and any shared pair leaves the core and the outermost group $\sqrt{2}$ apart at initialization and 2 apart in the update. $\square$

C.4 PROOF OF COROLLARY 1

Proof. The proof of Theorem 1 holds for every base width $g = md_0$ and every supported set $\mathcal{S} \subseteq \{1, \dots, M\}$. Its constants, $\Theta(\sigma_0^2 d_0)$ in Equation 16, $\Theta(\eta_0 d_0)$ in Equation 17 and the decay factor $1 - \eta_0 \lambda_0$ in Equation 18, depend only on the proxy width d_0 and on the tuned values: m and M enter only through the active input widths, which the rule of Table 1 cancels. The proxy is the special case $M = 1$, $g = d_0$ (so $m = s = 1$) of the same computation, so the values tuned on it set the same constants in every group and readout of every target. This gives (i) for any **SHELLELASTIC** supernet and (ii) under any increase of m or M . $\square$

C.5 EXTENSION TO OTHER ELASTIC ARCHITECTURES

The proofs above use the block structure of **SHELLELASTIC** only in Step 1 of Appendix C.2, to show that a group has the same fan-in and the same input in every sub-network that contains it. For the other elastic architectures of Section 3.2, under their rules in Appendix D, Theorem 1 holds with sub-network s in place of the group with boundary s : the hidden matrices of every sub-network satisfy (i), and (ii) at each step that trains that sub-network. The readouts are treated per architecture below.

StairFormer. Its hidden matrices have the connectivity of **SHELLELASTIC** with $\mathcal{S} = \{1, \dots, M\}$ and follow Table 1, and all sub-networks are trained at every step, so the proof of Theorem 1 applies verbatim. Its single readout, of which sub-network s reads the first sg columns, takes the split of Equation 23, which gives these columns the initialization $\sigma_0/\sqrt{ms}$ and the learning rate $\eta_0/(ms)$ of U_s in Table 1; since every sub-network feeds them the prefix $x_{:sg}^{(L)}$ up to its terminal norm, Steps 5 and 6 apply.

Fully elastic models. Sub-network s reads the top-left slice W_s of every hidden matrix, whose fan-in sg (esg for the FFN down projection) varies with s . By Equation 23, W_s has the initialization standard deviation $\sigma_0/\sqrt{ms}$ ($\sigma_0\sqrt{e}/\sqrt{ms}$ with the output multiplier $1/e$ for the FFN down projection), and since Δ moves by $\Theta(\eta_0/m)$ per coordinate under (A4), W_s moves by $\Theta(\eta_0/(ms))$ per coordinate. These are the values of Table 1 at fan-in sg , so Steps 2–4 give the pre-activations of sub-network s the scales $\Theta(\sigma_0\sqrt{d_0})$ at initialization and $\Theta(\eta_0 d_0)$ after one step. At a step that trains sub-network s , the update of Δ is computed from the input of sub-network s , so (A3) applies to the update of W_s ; (ii) does not need the steps that train other sub-networks. The shared readout takes the same split with the multiplier α_s (Appendix D), so the columns that sub-network s reads have the initialization $\sigma_0/\sqrt{ms}$ and the learning rate $\eta_0/(ms)$ of U_s in Table 1, and Step 5 gives its logits the same two scales.

MatFormer. Only the FFN down projection has a fan-in that varies with the sub-network, $sh_0 = esD_{\text{model}} = esmd_0$. With the split and the output multiplier $1/e$, its slice has the effective initialization $\sigma_0/\sqrt{ems}$ and the effective learning rate $\eta_0/(ems)$, so Step 4 gives the same constants with sD_{model} in place of sg . Every other hidden matrix and the shared readout have the fixed fan-in $D_{\text{model}} = md_0$ and take the factor m , which is the rule of Table 1 with $s = 1$ and gives the same constants by Steps 2, 3 and 5. As for fully elastic models, (A3) applies at each step to the sub-network trained at that step.

Transfer. In all three architectures, m and s enter only through fan-ins that the rules cancel, so the constants $\Theta(\sigma_0\sqrt{d_0})$ and $\Theta(\eta_0 d_0)$ and the decay factor $1 - \eta_0\lambda_0$, which Δ obtains from its learning rate η_0/m and weight decay $\lambda_0 m$, depend only on d_0 and the base values. The proof of Corollary 1 therefore carries over to every m and every supported set.

D μ -ELASTIC FOR OTHER ELASTIC ARCHITECTURES

All three elastic baselines use the proxy width $d_0 = 128$ and the base values $(\sigma_0, \eta_0, \lambda_0)$ of **SHELLELASTIC**, and their m and s follow Section 3.2. Tables 5 and 6 list the complete rules of the fully elastic model and MatFormer.

Fully elastic models. A fully elastic model stores every hidden matrix as one dense tensor at the full width, and sub-network s reads its top-left slice, whose fan-in sg (esg for the FFN down projection) varies with s . μ -ELASTIC gives this slice the factor ms : initialization $\sigma_0/\sqrt{ms}$, learning rate $\eta_0/(ms)$ and weight decay $\lambda_0 ms$. One stored tensor cannot follow this rule for every s , because the initialization and the update scale with different powers of s . We therefore split each hidden tensor into a frozen initialization W^{init} , drawn at $\sigma_0/\sqrt{m}$, and a trainable delta Δ , initialized to zero and trained with learning rate η_0/m and weight decay $\lambda_0 m$. Sub-network s uses

$$W_s = s^{-1/2} W^{\text{init}}[\text{slice}] + s^{-1} \Delta[\text{slice}]. \quad (23)$$

At initialization, W_s has standard deviation $s^{-1/2}\sigma_0/\sqrt{m} = \sigma_0/\sqrt{ms}$; under Adam, Δ moves by $\Theta(\eta_0/m)$ per coordinate, so W_s moves by $\Theta(\eta_0/(ms))$. Both match Table 1 for every s , while weight decay, which acts on Δ only, pulls W_s toward its initialization. The main results use one

Table 5: **μ -ELASTIC for the fully elastic baseline.** s is the sub-network width in units of g ; multipliers act on the sliced tensor in the forward pass (Equation 23), and the FFN down projection multiplies its output, from both terms, by $1/e$.

Parameter	Initialization	Learning rate	Weight decay	Forward multiplier
Token embedding (first sg columns)	σ_0	η_0	λ_0	1
W^{init} of attention and FFN up/gate	$\sigma_0/\sqrt{m}$	frozen	–	$s^{-1/2}$
W^{init} of FFN down	$\sigma_0\sqrt{e}/\sqrt{m}$	frozen	–	$s^{-1/2}$
Δ of every hidden matrix	0	η_0/m	$\lambda_0 m$	s^{-1}
W^{init} of the readout (first sg columns)	$\sigma_0/\sqrt{m}$	frozen	–	$\alpha_s s^{-1/2}$
Δ of the readout (first sg columns)	0	η_0/m	$\lambda_0 m$	$\alpha_s s^{-1}$
Norm gains, readout scalars α_s	1	η_0	0	–

Table 6: **μ -ELASTIC for MatFormer.** $s = D_{\text{ffn}}^{(s)}/h_0$ is the FFN hidden width in units of the base FFN width h_0 ; the hidden matrices other than the FFN down projection and the readout have a fixed fan-in and take the non-elastic μ P factor $m = D_{\text{model}}/d_0$, and the embedding and gains keep the base values. The FFN down projection multiplies its output, from both terms, by $1/e$.

Parameter	Initialization	Learning rate	Weight decay	Forward multiplier
Token embedding (full width)	σ_0	η_0	λ_0	1
W^{init} of attention and FFN up/gate	$\sigma_0/\sqrt{m}$	frozen	–	1
Δ of attention and FFN up/gate	0	η_0/m	$\lambda_0 m$	1
W^{init} of FFN down (first $D_{\text{ffn}}^{(s)}$ columns)	$\sigma_0\sqrt{e}/\sqrt{m}$	frozen	–	$s^{-1/2}$
Δ of FFN down	0	η_0/m	$\lambda_0 m$	s^{-1}
Shared readout (fan-in D_{model})	$\sigma_0/\sqrt{m}$	η_0/m	$\lambda_0 m$	α (one scalar)
Norm gains, readout scalar α	1	η_0	0	–

shared sliced readout, of which sub-network s reads the first sg columns; it takes the same split with the multiplier α_s (Table 5), which gives these columns the initialization $\sigma_0/\sqrt{ms}$ and the learning rate $\eta_0/(ms)$ of the per-width readout U_s of Table 1. Our baseline has $g = 64$ ($m = 1/2$), $\mathcal{S} = \{6, 7, 8, 9, 10\}$ and $e = 4$; attention nests by heads at $d_{\text{head}} = 64$, the norms act over the active width, the embedding is sliced, and one sub-network is sampled uniformly at random per step.

MatFormer. MatFormer keeps D_{model} fixed, so $m = D_{\text{model}}/d_0$, and nests only the FFN hidden width, $D_{\text{ffn}}^{(s)} = s h_0$ with a base FFN width h_0 ; here $e = h_0/D_{\text{model}}$ takes the place of the FFN expansion, so that the down projection of sub-network s reads esD_{model} coordinates. Only the FFN down projection has a fan-in that varies across sub-networks; it takes the factor ms through the split of Equation 23, and the other hidden matrices and the readout, whose fan-in D_{model} is fixed, take the non-elastic μ P factor m , the attention and FFN up/gate matrices through the same split with multiplier 1; the embedding and gains keep the base values (Table 6). Our baseline has $D_{\text{model}} = 512$ ($m = 4$), $h_0 = 128$ ($e = 1/4$) and $\mathcal{S} = \{4, 9, 15, 22, 30\}$; attention, normalization, the embedding and a single readout are shared at full width, and one sub-network is sampled uniformly at random per step, as in the original MatFormer recipe.

StairFormer. With equal-width nested blocks, StairFormer has the connectivity of **SHELLELASTIC** with $\mathcal{S} = \{1, \dots, M\}$, so its hidden matrices follow Table 1 without a split. Its single readout, of which sub-network s reads the first sg columns, has a fan-in that varies with s and takes the factor ms through the split of Equation 23. Our StairFormer baseline has five 208-wide groups with $d_{\text{head}} = 104$, so that each group holds two complete heads.

E TRAINING DETAILS

This appendix gives the model configurations of the three scales (Appendix E.1), the optimization settings (Appendix E.2), the parameter counts (Appendix E.3), and the training and inference cost (Appendix E.4).

E.1 MODEL CONFIGURATIONS

Table 7 lists the base width, supported set and size of every model, and Table 8 the remaining architecture settings; Figure 7 shows one layer of **SHELLELASTIC** and Figure 6 the complete model. Ablations vary only the setting under study relative to the defaults of the 119M–285M scale.

119M–285M scale. The main results at this scale use the models of the first block of Table 7, trained on 6.55B DCLM-Baseline tokens with the HPs of Table 9. **SHELLELASTIC** has $g = 128$ with one head per base width ($d_{\text{head}} = 128$), $M = 8$ and $\mathcal{S} = \{4, 5, 6, 7, 8\}$, so each hidden matrix keeps 42 of its 64 blocks of size 128×128 , and the FFN expansion is $e = 1$. The two branch norms of every layer are group-local over $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4, \mathcal{G}_5$ of widths (512, 128, 128, 128, 128), queries and keys are normalized per head, and each supported width has its own terminal norm N_s^{out} over all sg coordinates, readout U_s and learned scalar α_s ; all five widths are selected at every step with loss weights (1, 1, 1, 1, 4). The proxy is the non-elastic special case $M = 1$ at width $d_0 = 128$, and each standalone model is a separately trained model of width 384 to 640 with FFN expansion 4 and $d_{\text{head}} = 64$; the sampled standalone baseline trains these five models in one run of the default length, updating one model sampled uniformly at random per step. Since each of these models takes one step in five on average, the sampled standalone baseline rescales their optimizer HPs to this five times shorter horizon with the token-budget rule of Mlodozieniec et al. (2026), consistent with the scaling rules of Section 3.2.

375M–906M scale. The main results at this scale use the models of the second block of Table 7. **SHELLELASTIC** has $g = 256$ with two heads per base width ($d_{\text{head}} = 128$, $m = 2$) and otherwise the settings above: $M = 8$, $\mathcal{S} = \{4, 5, 6, 7, 8\}$, $e = 1$, groups $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4, \mathcal{G}_5$ of widths (1024, 256, 256, 256, 256) and loss weights (1, 1, 1, 1, 4). MatFormer has $D_{\text{model}} = 1024$ ($m = 8$), $h_0 = 128$ ($e = 1/8$) and $\mathcal{S} = \{8, 17, 27, 38, 51\}$, and the fully elastic model has $g = 64$ ($m = 1/2$), $\mathcal{S} = \{13, 15, 17, 19, 21\}$ and $e = 2.75$. The standalone models have widths 832 to 1344 ($g = 64$, $M = 13, 15, 17, 19, 21$) with FFN expansion 2.75 and $d_{\text{head}} = 64$, and the sampled standalone baseline trains these five models in one run. Every run at this scale takes 50,000 steps of 256 sequences (26.2B tokens), decays the learning rate linearly from step 40,000 to 0 at step 50,000 and reports the validation loss at step 50,000; all other settings are those of the 119M–285M scale.

675M–1.69B scale. The main results at this scale use the models of the third block of Table 7. **SHELLELASTIC** is the supernet of the 375M–906M scale with FFN expansion $e = 4$. MatFormer has $D_{\text{model}} = 1024$ ($m = 8$), $h_0 = 256$ ($e = 1/4$) and $\mathcal{S} = \{16, 24, 34, 45, 57\}$, and the fully elastic model and the standalone models keep the widths of the 375M–906M scale with FFN expansion 7.25. Every run at this scale takes 100,000 steps of 256 sequences (52.4B tokens), decays the learning rate linearly from step 85,000 to 0 at step 100,000 and reports the validation loss at step 100,000. For the doubled horizon, the optimizer HPs are rescaled to $\eta_0 = 1.41 \times 10^{-3}$, $\lambda_0 = 7.07 \times 10^{-3}$, $\beta_1 = \beta_2 = 0.975$ and $\epsilon = 1.41 \times 10^{-14}$, and the sampled standalone baseline rescales its own values in the same way; each model is fully sharded over the 8 GPUs, and all other settings are those of the 375M–906M scale.

E.2 OPTIMIZATION

Table 9 lists the optimization settings used unless otherwise specified: the proxy sweeps, the elastic models, the baselines and the standalone models differ only in their architecture and in the per-group multipliers of their parameterization. The training objective includes the z -loss of Equation 8 with $\lambda_z = 10^{-4}$; all reported losses, both training and validation, are the cross-entropy alone, averaged over non-padded tokens.

E.3 PARAMETER COUNTS

The parameters of a sub-network count the non-zero weights it reads, its embedding columns, and its own terminal norm and readout; the two tensors of a frozen-plus-delta split (Equation 23) count once. The baselines were sized to these counts (Table 8), and Table 10 lists them for every sub-network of the main results at the three scales.

Table 7: **Model configurations.** Each model is defined by its base width g and supported set $\mathcal{S}$ (Section 3.1): its sub-networks have widths sg for $s \in \mathcal{S}$, and a non-elastic model has $\mathcal{S} = \{M\}$. MatFormer nests only the FFN hidden width, so its g and $\mathcal{S}$ refer to that width at a fixed D_{model} (512, and 1024 at the two larger scales). All models have 32 layers and a 50,432-token vocabulary; parameters count the non-zero weights of each sub-network (Appendix E.3).

Model	g	$\mathcal{S}$	Sub-network widths	d_{head}	D_{ffn}	Parameters (M)
Proxy	128	$\{1\}$	128	128	128	17.1
<i>119M–285M scale</i>						
Standalone	64	$\{M\}, M = 6, 7, 8, 9, 10$	384–640	64	$4D_{\text{model}}$	119.0–287.5
MatFormer	128	$\{4, 9, 15, 22, 30\}$	FFN 512–3840	128	3840	118.8–282.4
Fully elastic	64	$\{6, 7, 8, 9, 10\}$	384–640	64	2560	119.0–287.5
SHELLELASTIC	128	$\{4, 5, 6, 7, 8\}$	512–1024	128	1024	118.8–279.6
<i>375M–906M scale</i>						
Standalone	64	$\{M\}, M = 13, 15, 17, 19, 21$	832–1344	64	$2.75D_{\text{model}}$	377.5–901.6
MatFormer	128	$\{8, 17, 27, 38, 51\}$	FFN 1024–6528	128	6528	371.9–912.9
Fully elastic	64	$\{13, 15, 17, 19, 21\}$	832–1344	64	3696	377.5–901.6
SHELLELASTIC	256	$\{4, 5, 6, 7, 8\}$	1024–2048	128	2048	371.9–911.5
<i>675M–1.69B scale</i>						
Standalone	64	$\{M\}, M = 13, 15, 17, 19, 21$	832–1344	64	$7.25D_{\text{model}}$	676.6–1682.0
MatFormer	256	$\{16, 24, 34, 45, 57\}$	FFN 4096–14592	128	14592	673.8–1705.6
Fully elastic	64	$\{13, 15, 17, 19, 21\}$	832–1344	64	9744	676.6–1682.0
SHELLELASTIC	256	$\{4, 5, 6, 7, 8\}$	1024–2048	128	8192	673.8–1704.2

Table 8: **Architecture settings at the 119M–285M scale.** The upper block is shared by all models, and the lower block lists the settings that differ between them (base widths, supported sets and sizes of all three scales in Table 7).

<i>Shared by all models</i>					
Layers	32 pre-norm layers with residual-branch multiplier 1				
Normalization	RMSNorm with learnable gains and $\epsilon_N = 10^{-5}$; query–key RMSNorm per head				
Attention	rotary embeddings with base 10,000, logit scale $d_{\text{head}}^{-1/2}$, elementwise sigmoid gate				
FFN and biases	SwiGLU; no biases				
Embeddings	untied, over the 50,432-token GPT-NeoX vocabulary				
Initialization	truncated normal at ± 3 standard deviations				
Setting	Proxy	Standalone	MatFormer	Fully elastic	SHELLELASTIC
Elastic dimensions	–	–	FFN hidden width	residual, heads, FFN hidden width, embedding, readout	residual, heads, FFN hidden width, embedding, terminal norm, readout
Trunk normalization	RMSNorm	RMSNorm	RMSNorm	over the active width	group-local over $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4, \mathcal{G}_5$
Terminal norm and readout	single	single	shared, one learned α	shared and sliced, learned α_s	per width: N_s^{out}, U_s , learned α_s
Sub-networks per step	–	–	one, uniformly at random	one, uniformly at random	all five, $\omega_s = (1, 1, 1, 1, 4)$
Parameterization	$\mu\text{P}, m = 1$	$\mu\text{P}, m = \text{width}/128$	$\mu\text{-ELASTIC}, m = 4$ (Table 6)	$\mu\text{-ELASTIC}, m = 1/2$ (Table 5)	$\mu\text{-ELASTIC}, m = 1$ (Table 1)
Parameters (M)	17.1	119.0–287.5	118.8–282.4	119.0–287.5	118.8–279.6

E.4 TRAINING AND INFERENCE COST

Compute. By Equation 9, a step of **SHELLELASTIC** runs one trunk pass at the full width and adds only one terminal norm and readout per supported width. Updating every width once takes one trunk

Table 9: **Training HPs** used unless otherwise specified (Appendix E.1 lists the changes at the two larger scales). The base values η_0 , σ_0 and λ_0 are scaled per parameter group by μ -ELASTIC (Table 1 and Appendix D) or by μ P; SP runs use them verbatim.

HP	Value
Sequence length	2,048
Batch size	128 sequences (262,144 tokens)
Training steps	25,000 (6.55B tokens)
Learning-rate schedule	warmup-stable-decay
Learning-rate warmup	linear, first 1,000 steps
Learning-rate decay	linear to 0, from step 20,000 to step 25,000
Base learning rate η_0	2×10^{-3}
Base initialization σ_0	3.3×10^{-3} , truncated normal at ± 3 standard deviations
Optimizer	AdamW with decoupled weight decay
Base weight decay λ_0	0.01; none on norm gains and α_s
β_1, β_2	0.95, 0.95
AdamW ϵ	10^{-14}
Gradient clipping	global norm 4
Step skipping	after step 5,000, a step is skipped if its gradient norm exceeds $20\times$ the clip value
z -loss coefficient λ_z	10^{-4}
Precision	bf16 mixed precision with fp32 master weights and fp32 gradient reduction
Parallelism	hybrid-sharded data parallelism (4-GPU shards, two replicas)
Hardware	one node of 8 A100, H100 or H200 GPUs
Seeds	95 (model), 1 (data)
Validation	C4 validation loss every 2,500 steps, reported at step 25,000

Table 10: **Parameters (M) of the sub-networks of Table 2** at each scale and tier, counted as in Appendix E.3; Avg. averages the five tiers. Standalone, sampled standalone and the fully elastic model share the same five dense architectures.

	Method	Low	Medium	High	Extra	Max	Avg.
119M–285M	Standalone	119.0	154.4	194.3	238.7	287.5	198.8
	Sampled standalone	119.0	154.4	194.3	238.7	287.5	198.8
	MatFormer	118.8	150.3	188.0	232.1	282.4	194.3
	Fully elastic	119.0	154.4	194.3	238.7	287.5	198.8
	SHELLELASTIC	118.8	152.7	190.8	233.1	279.6	195.0
375M–906M	Standalone	377.5	487.7	611.8	749.8	901.6	625.7
	Sampled standalone	377.5	487.7	611.8	749.8	901.6	625.7
	MatFormer	371.9	485.1	610.9	749.3	912.9	626.0
	Fully elastic	377.5	487.7	611.8	749.8	901.6	625.7
	SHELLELASTIC	371.9	481.6	608.1	751.4	911.5	624.9
675M–1.69B	Standalone	676.6	885.8	1123.2	1388.5	1682.0	1151.2
	Sampled standalone	676.6	885.8	1123.2	1388.5	1682.0	1151.2
	MatFormer	673.8	875.2	1126.8	1403.7	1705.6	1157.0
	Fully elastic	676.6	885.8	1123.2	1388.5	1682.0	1151.2
	SHELLELASTIC	673.8	878.0	1117.7	1393.1	1704.2	1153.4

pass per width with MatFormer, the fully elastic model and the standalone models, and sampling one width per step instead makes the cost vary from step to step.

Measurement protocol. For Table 11, each tier runs as its own sub-network on A100-40GB GPUs, with the model sharded as in training. Latency is the mean of three passes after two warm-up passes (forward and backward for training, forward only for inference), and peak training memory is PyTorch’s peak allocated memory per GPU during a training step, which for **SHELLELASTIC** trains all five tiers.

Table 11: **Latency and peak training memory at the 119M–285M scale** on A100-40GB GPUs, for one GPU’s batch of 16 sequences of 2,048 tokens, with each tier run as its own sub-network (protocol in Appendix E.4). Entries are means over runs, with the standard deviation as a subscript; Avg. averages the tiers. [†]Five separately trained models.

Method	Tier 1 Low	Tier 2 Medium	Tier 3 High	Tier 4 Extra	Tier 5 Max	Avg.
<i>Train latency (ms, forward + backward)</i>						
Standalone [†]	570 _{±2}	644 _{±3}	723 _{±6}	834 _{±5}	917 _{±10}	737 _{±3}
Sampled standalone	<i>as Standalone</i>					
MatFormer	641 _{±1}	677 _{±2}	722 _{±4}	776 _{±5}	834 _{±6}	730 _{±4}
Fully elastic	597 _{±4}	671 _{±3}	748 _{±5}	858 _{±7}	933 _{±9}	761 _{±6}
SHELLELASTIC	437 _{±11}	527 _{±13}	603 _{±8}	690 _{±2}	747 _{±3}	601 _{±6}
<i>Inference latency (ms, forward)</i>						
Standalone [†]	151 _{±2}	175 _{±1}	199 _{±2}	228 _{±2}	254 _{±2}	201 _{±1}
Sampled standalone	<i>as Standalone</i>					
MatFormer	175 _{±1}	186 _{±1}	200 _{±1}	216 _{±2}	235 _{±2}	202 _{±1}
Fully elastic	160 _{±2}	183 _{±1}	207 _{±1}	236 _{±1}	260 _{±1}	209 _{±1}
SHELLELASTIC	141 _{±10}	167 _{±15}	178 _{±12}	205 _{±1}	218 _{±2}	182 _{±8}
<i>Peak training memory (GB per GPU)</i>						
Standalone [†]	15.0 _{±.0}	17.5 _{±.0}	20.1 _{±.0}	22.6 _{±.0}	25.2 _{±.0}	20.1 _{±.0}
Sampled standalone	<i>as Standalone</i>					
MatFormer	15.8 _{±.0}	17.9 _{±.0}	20.3 _{±.0}	23.2 _{±.0}	26.5 _{±.0}	20.7 _{±.0}
Fully elastic	16.0 _{±.0}	18.5 _{±.0}	20.9 _{±.0}	23.4 _{±.0}	25.8 _{±.0}	20.9 _{±.0}
SHELLELASTIC	21.6 _{±.0} (a step that trains all five tiers)					–

Activation checkpointing. The runs of Table 11 train with activation checkpointing on every other transformer block (16 of 32), whose forward pass is recomputed during the backward pass. Training latency therefore includes this recomputation (on H100 GPUs, the Standalone models train 14–19% faster without it), and peak training memory keeps only the inputs of the checkpointed blocks (without checkpointing, Standalone needs $1.9\times$ the memory); inference latency is a plain forward pass and is unaffected.

Sampled standalone. Sampled standalone trains the same five models as Standalone, one per step, so its per-tier latency and memory are those of Standalone: in its training runs, the logged latencies agree within 2%, and the 2.2–2.6 GB of extra memory it logged comes from our implementation keeping the four inactive models’ weights and optimizer states on the GPU, which offloading them would remove.

Observations. Two observations emerge:

- **The shared trunk is the saving of the architecture.** One training step of **SHELLELASTIC** updates all five tiers, whereas updating every tier once takes five steps with MatFormer and the fully elastic model (one per tier) and one step of each model with Standalone.
- **Fused kernels lower the per-tier cost.** At equal parameters, all methods need about the same effective FLOPs per tier, so the lower per-tier latency and memory of **SHELLELASTIC** in Table 11 come from our fused implementation: Triton kernels compute the RMSNorms of every transformer block, including the query and key normalizations, and read the queries and keys directly from the output of the QKV projection without copying them.

F EVALUATION DETAILS

Validation loss. The validation loss is the next-token cross-entropy on held-out C4 text, averaged over non-padded tokens, computed every 2,500 steps and reported at the final step. Each supported

sub-network is extracted from the supernet as in Section 3.3 and evaluated on its own, on the same validation batches as every other model, so losses are comparable across widths, families and the standalone models.

Downstream tasks. During pretraining, we also probe 5-shot accuracy on ARC-Easy, PIQA and HellaSwag at the same interval. After pretraining, we evaluate every sub-network on 13 datasets spanning five skill areas, following the protocol of DCLM and OLMoE, in a 5-shot setting on the standard community splits (Table 12). Multiple-choice tasks are scored by accuracy, with a random baseline of 25% for four-choice and 50% for two-choice items; WinoGrande uses its cloze formulation, LAMBADA is scored by exact match of the final word, and SVAMP by exact match of the generated numerical answer. For elastic models, we report the accuracy of every sub-network and their mean. The datasets of each skill area are described below.

Table 12: **Downstream evaluation suite.** All tasks are evaluated 5-shot; ARC-Easy, PIQA and HellaSwag are also probed during pretraining. MC: multiple-choice accuracy; EM: exact match.

Field	Dataset	Split	#Items	Metric	Random
World knowledge	MMLU	test	14,042	MC accuracy	25%
	ARC-Easy	test	2,376	MC accuracy	25%
	ARC-Challenge	test	1,172	MC accuracy	25%
Symbolic problem solving	SVAMP	test	300	EM	0%
Commonsense reasoning	COPA	val	100	MC accuracy	50%
	PIQA	val	1,838	MC accuracy	50%
Language understanding	HellaSwag	val	10,042	MC accuracy	25%
	WinoGrande	val	1,267	cloze accuracy	50%
	LAMBADA	test	5,153	final-word EM	0%
Reading comprehension	BoolQ	val	3,270	MC accuracy	50%
	AGIEval LSAT-RC	test	268	MC accuracy	25%
	AGIEval LSAT-LR	test	510	MC accuracy	25%
	AGIEval SAT-EN	test	206	MC accuracy	25%

World knowledge. Exam-style and science questions that probe academic knowledge.

- **MMLU** (Hendrycks et al., 2020) (14,042 items, four-choice): exam-style questions from 57 subjects across the sciences, humanities and social sciences.
- **ARC-Easy** and **ARC-Challenge** (Clark et al., 2018) (2,376 and 1,172 items, four-choice): grade-school science questions; the Challenge subset holds the questions that retrieval and co-occurrence methods answer incorrectly.

Symbolic problem solving. Arithmetic reasoning with a generated answer.

- **SVAMP** (Patel et al., 2021) (300 problems, free-response): short arithmetic word problems with numerical answers; the model generates a derivation followed by the answer, which is scored by exact match.

Commonsense reasoning. Everyday causal and physical knowledge.

- **COPA** (Gordon et al., 2012) (100 items, two-choice): choose the more plausible cause or effect of a premise.
- **PIQA** (Bisk et al., 2020) (1,838 items, two-choice): choose the more sensible solution to a physical task.

Language understanding. Narrative coherence, coreference and the use of broad context.

- **HellaSwag** (Zellers et al., 2019) (10,042 items, four-choice): choose the most plausible continuation of a partial narrative.

- **WinoGrande** (Sakaguchi et al., 2021) (1,267 items, two-choice): fill a blank with one of two candidates so that the sentence is coherent.
- **LAMBADA** (Paperno et al., 2016) (5,153 passages): predict the final word of a passage that requires broad discourse context.

Reading comprehension. Questions answered from a given passage or argument.

- **BoolQ** (Clark et al., 2019) (3,270 items, yes/no): answer a binary question about a short passage.
- **AGIEval LSAT-RC, LSAT-LR and SAT-EN** (Zhong et al., 2024) (268, 510 and 206 items, four-choice): LSAT reading-comprehension and logical-reasoning questions and SAT English questions.

G COMPLETE MAIN RESULTS

Tables 13–27 give every metric of every sub-network in Table 2, one table per scale and tier, and Figures 9–11 show the training curves of each scale:

- **119M–285M scale** (25,000 steps, three runs): Figure 9 and Tables 13–17;
- **375M–906M scale** (50,000 steps, three runs): Figure 10 and Tables 18–22;
- **675M–1.69B scale** (100,000 steps, single runs): Figure 11 and Tables 23–27.

Tables. Each table gives the parameter count, the train loss (cross-entropy without the z -loss, mean over the last 500 steps), the validation loss at the last step, and the 5-shot accuracy (%) on each of the 13 downstream tasks of Appendix F, grouped by field as there (LSAT-RC, LSAT-LR and SAT-EN are from AGIEval), with their average (the accuracy of Table 2). At the 119M–285M and 375M–906M scales, entries are the mean over three runs of the same configuration and seed, with the standard deviation as a subscript; at the 675M–1.69B scale, every entry is a single run. Standalone[†] is five separately trained models, each trained for the full number of steps, which together take $5 \times$ the optimizer steps of any other column. Bold marks the best entry among the four methods that train all five sizes within one run.

Curves. Each figure has one column per tier and plots, against the global training step, the train loss (the logged means over 50-step intervals), the validation loss, and 5-shot accuracy on ARC-Easy, HellaSwag and PIQA, the three benchmarks evaluated during training. Sampled standalone, MatFormer and the fully elastic model sample one sub-network uniformly at random per step, **SHELLELASTIC** updates all sizes at every step, and Standalone trains each size separately for the full number of steps ($5 \times$ the steps). Values are plotted as logged, without smoothing or clipping, and y-axes are shared within each row.

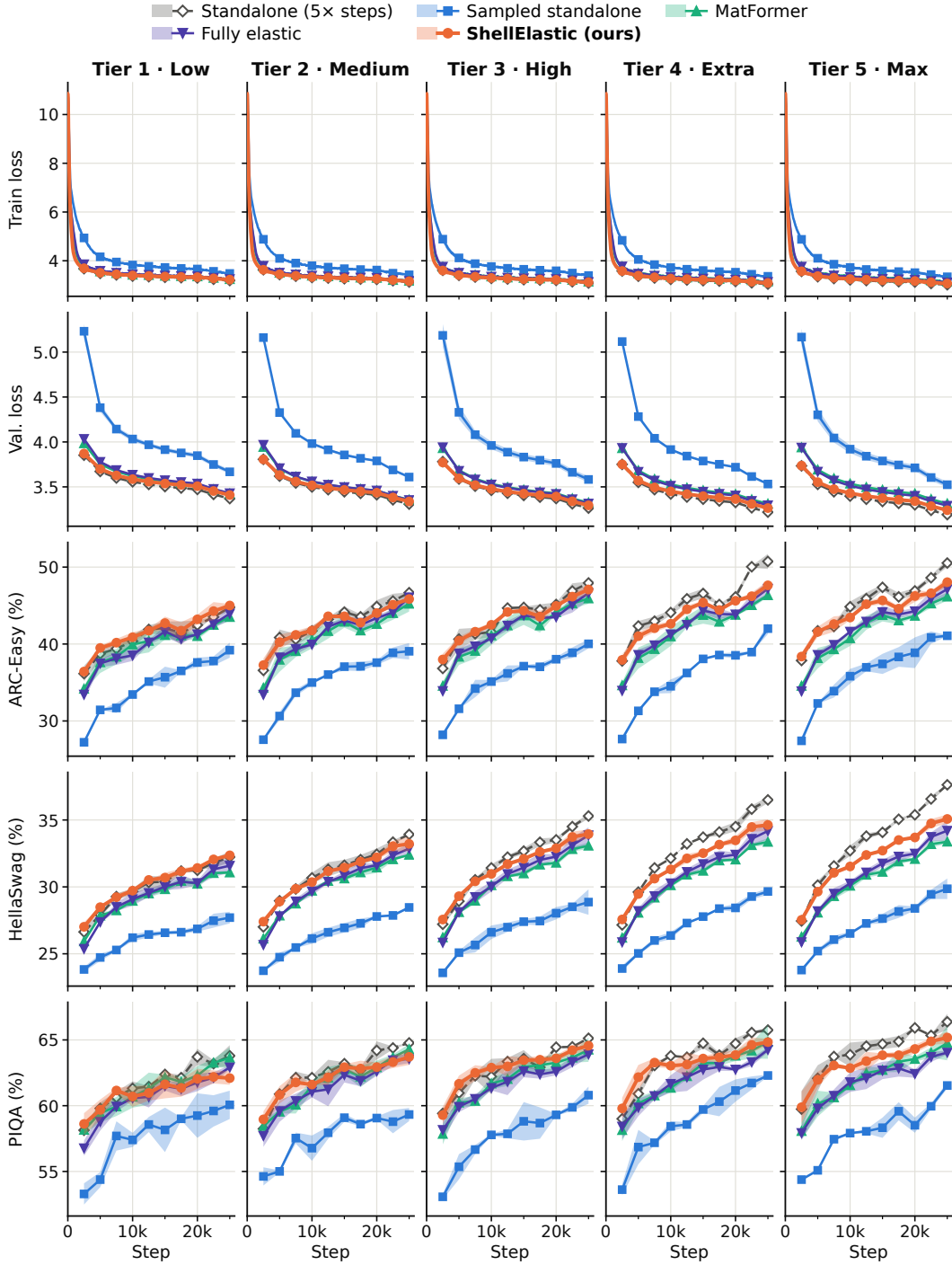


Figure 9: **Training curves at the 119M–285M scale** (Table 2). Columns are the five tiers, from Tier 1 (Low, about 119M parameters) to Tier 5 (Max, about 285M); rows are the train loss, the validation loss and 5-shot accuracy on ARC-Easy, HellaSwag and PIQA. Lines are means over three runs, with shaded bands of ± 1 standard deviation; conventions in Appendix G.

Table 13: **Per-task results at the 119M–285M scale, Tier 1 (Low, about 119M parameters).** Sub-network width 384 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 512 at $D_{\text{model}} = 512$ for MatFormer, and 512 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 1 · Low	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	119.0	119.0	118.8	119.0	118.8
Train loss ↓	3.1951 $\pm$.0035	3.4911 $\pm$.0235	3.2536 $\pm$.0032	3.2572 $\pm$.0091	3.2305 $\pm$.0015
Val. loss ↓	3.3695 $\pm$.0034	3.6674 $\pm$.0242	3.4248 $\pm$.0036	3.4291 $\pm$.0097	3.4086 $\pm$.0009
<i>5-shot accuracy (%) ↑</i>					
MMLU	24.5 $\pm$.3	24.4 $\pm$.2	24.8 $\pm$.5	24.7 $\pm$.7	24.7 $\pm$.6
ARC-Easy	44.8 $\pm$.5	39.2 $\pm$ 1.0	43.5 $\pm$.5	44.0 $\pm$.9	45.0 $\pm$.2
ARC-Challenge	23.3 $\pm$ 1.7	22.7 $\pm$.5	23.9 $\pm$.7	23.3 $\pm$ 1.2	23.3 $\pm$.5
SVAMP	5.7 $\pm$.5	6.3 $\pm$.9	6.8 $\pm$ 1.5	6.4 $\pm$.8	5.7 $\pm$.3
COPA	61.0 $\pm$.0	54.0 $\pm$ 5.7	57.7 $\pm$.6	58.7 $\pm$ 1.5	59.7 $\pm$.6
PIQA	63.8 $\pm$.8	60.1 $\pm$ 1.1	63.6 $\pm$.8	62.9 $\pm$.2	62.1 $\pm$.5
HellaSwag	32.2 $\pm$.4	27.7 $\pm$.4	31.1 $\pm$.1	31.6 $\pm$.5	32.4 $\pm$.2
WinoGrande	52.5 $\pm$.1	52.1 $\pm$.2	50.7 $\pm$.8	49.0 $\pm$ 1.3	50.7 $\pm$.8
LAMBADA	25.7 $\pm$ 1.1	17.4 $\pm$ 1.7	24.7 $\pm$.4	23.6 $\pm$ 1.0	25.9 $\pm$.7
BoolQ	51.1 $\pm$ 7.2	54.9 $\pm$ 7.6	50.4 $\pm$ 5.8	53.0 $\pm$.6	55.1 $\pm$ 7.2
LSAT-RC	25.2 $\pm$ 1.3	25.2 $\pm$.3	24.3 $\pm$ 1.7	25.1 $\pm$ 1.8	24.8 $\pm$ 1.8
LSAT-LR	26.9 $\pm$.0	28.8 $\pm$.3	27.9 $\pm$.6	25.4 $\pm$ 1.3	26.2 $\pm$ 1.0
SAT-EN	26.5 $\pm$ 1.7	26.9 $\pm$.3	26.7 $\pm$ 1.3	26.1 $\pm$ 1.7	25.6 $\pm$.3
Average	35.6 $\pm$.9	33.8 $\pm$.2	35.1 $\pm$.3	34.9 $\pm$.2	35.5 $\pm$.4

Table 14: **Per-task results at the 119M–285M scale, Tier 2 (Medium, about 153M parameters).** Sub-network width 448 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 1152 at $D_{\text{model}} = 512$ for MatFormer, and 640 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 2 · Medium	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	154.4	154.4	150.3	154.4	152.7
Train loss ↓	3.1344 $\pm$.0019	3.4301 $\pm$.0095	3.1821 $\pm$.0034	3.1783 $\pm$.0083	3.1641 $\pm$.0031
Val. loss ↓	3.3081 $\pm$.0016	3.6091 $\pm$.0104	3.3566 $\pm$.0037	3.3536 $\pm$.0083	3.3375 $\pm$.0027
<i>5-shot accuracy (%) ↑</i>					
MMLU	25.0 $\pm$.4	24.9 $\pm$.3	24.9 $\pm$.2	24.7 $\pm$.3	23.8 $\pm$.7
ARC-Easy	46.7 $\pm$.1	39.1 $\pm$ 1.1	45.2 $\pm$ 1.0	46.1 $\pm$.8	45.8 $\pm$.1
ARC-Challenge	24.4 $\pm$.4	23.0 $\pm$.7	23.6 $\pm$.0	24.0 $\pm$.4	23.5 $\pm$.4
SVAMP	6.8 $\pm$.2	5.8 $\pm$.7	6.7 $\pm$ 1.5	7.2 $\pm$ 1.4	6.3 $\pm$.9
COPA	59.0 $\pm$ 1.4	58.0 $\pm$ 1.4	60.0 $\pm$ 1.7	60.3 $\pm$ 1.5	61.7 $\pm$ 1.2
PIQA	64.8 $\pm$.0	59.3 $\pm$.5	64.3 $\pm$.4	63.5 $\pm$.3	63.7 $\pm$.6
HellaSwag	33.9 $\pm$.1	28.5 $\pm$.1	32.4 $\pm$.1	32.9 $\pm$.3	33.2 $\pm$.5
WinoGrande	51.7 $\pm$ 2.2	51.5 $\pm$ 2.7	50.5 $\pm$.6	50.1 $\pm$.3	49.3 $\pm$ 1.8
LAMBADA	28.4 $\pm$.3	18.9 $\pm$.3	26.4 $\pm$.6	26.4 $\pm$ 1.2	29.1 $\pm$.7
BoolQ	49.9 $\pm$ 4.8	48.1 $\pm$ 6.2	50.3 $\pm$ 6.1	50.7 $\pm$.6	57.2 $\pm$ 3.4
LSAT-RC	24.6 $\pm$ 4.2	24.8 $\pm$ 2.9	22.0 $\pm$ 3.3	24.6 $\pm$.4	25.6 $\pm$ 1.4
LSAT-LR	24.9 $\pm$.6	26.3 $\pm$ 1.9	25.9 $\pm$ 1.5	26.3 $\pm$ 3.0	29.2 $\pm$.9
SAT-EN	25.7 $\pm$.0	25.5 $\pm$ 1.7	25.2 $\pm$ 3.0	24.8 $\pm$ 1.3	27.2 $\pm$ 3.0
Average	35.8 $\pm$ 1.0	33.4 $\pm$ 1.1	35.2 $\pm$.2	35.5 $\pm$.4	36.6 $\pm$.6

Table 15: **Per-task results at the 119M–285M scale, Tier 3 (High, about 192M parameters).** Sub-network width 512 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 1920 at $D_{\text{model}} = 512$ for MatFormer, and 768 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 3 · High	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	194.3	194.3	188.0	194.3	190.8
Train loss ↓	3.0916 $\pm$.0086	3.4087 $\pm$.0336	3.1561 $\pm$.0036	3.1447 $\pm$.0077	3.1229 $\pm$.0018
Val. loss ↓	3.2635 $\pm$.0092	3.5826 $\pm$.0344	3.3250 $\pm$.0036	3.3145 $\pm$.0076	3.2944 $\pm$.0016
<i>5-shot accuracy (%) ↑</i>					
MMLU	24.6 $\pm$.4	25.0 $\pm$.3	24.6 $\pm$.4	24.6 $\pm$.5	24.0 $\pm$.5
ARC-Easy	47.9 $\pm$.2	40.0 $\pm$.6	45.9 $\pm$.7	46.6 $\pm$ 1.0	47.1 $\pm$.8
ARC-Challenge	23.9 $\pm$.8	22.3 $\pm$.8	24.3 $\pm$.3	24.5 $\pm$.7	23.6 $\pm$.8
SVAMP	7.7 $\pm$ 1.9	6.3 $\pm$.0	6.6 $\pm$ 1.5	6.8 $\pm$ 3.3	6.3 $\pm$ 1.0
COPA	60.5 $\pm$ 2.1	55.0 $\pm$ 1.4	60.3 $\pm$ 2.3	60.3 $\pm$ 3.1	61.7 $\pm$ 3.1
PIQA	65.1 $\pm$.1	60.8 $\pm$.7	64.3 $\pm$.9	63.9 $\pm$.3	64.6 $\pm$.4
HellaSwag	35.3 $\pm$.2	28.9 $\pm$ 1.0	33.1 $\pm$.2	33.9 $\pm$.6	33.9 $\pm$.3
WinoGrande	52.0 $\pm$.6	50.9 $\pm$.6	50.5 $\pm$.8	50.2 $\pm$.5	50.0 $\pm$ 2.1
LAMBADA	29.6 $\pm$.1	19.3 $\pm$ 1.0	27.6 $\pm$.4	28.0 $\pm$ 1.4	29.9 $\pm$ 1.1
BoolQ	54.1 $\pm$.1	53.9 $\pm$.8	51.1 $\pm$ 7.6	50.1 $\pm$ 3.5	55.3 $\pm$ 1.4
LSAT-RC	25.6 $\pm$.3	26.3 $\pm$ 2.4	23.6 $\pm$ 1.3	24.0 $\pm$ 1.6	26.0 $\pm$ 1.5
LSAT-LR	24.9 $\pm$.6	27.1 $\pm$ 1.7	27.1 $\pm$ 1.2	25.7 $\pm$ 2.2	26.8 $\pm$ 1.6
SAT-EN	24.8 $\pm$.7	26.5 $\pm$ 1.0	26.5 $\pm$ 3.0	25.2 $\pm$ 1.5	26.2 $\pm$ 1.7
Average	36.6 $\pm$.2	34.0 $\pm$.7	35.8 $\pm$.5	35.7 $\pm$.4	36.6 $\pm$.4

Table 16: **Per-task results at the 119M–285M scale, Tier 4 (Extra, about 236M parameters).** Sub-network width 576 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 2816 at $D_{\text{model}} = 512$ for MatFormer, and 896 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 4 · Extra	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	238.7	238.7	232.1	238.7	233.1
Train loss ↓	3.0474 $\pm$.0006	3.3561 $\pm$.0056	3.1407 $\pm$.0036	3.1233 $\pm$.0071	3.0941 $\pm$.0015
Val. loss ↓	3.2186 $\pm$.0012	3.5319 $\pm$.0056	3.3123 $\pm$.0037	3.2954 $\pm$.0072	3.2641 $\pm$.0017
<i>5-shot accuracy (%) ↑</i>					
MMLU	24.6 $\pm$.5	24.7 $\pm$.1	24.5 $\pm$.4	24.6 $\pm$.5	24.4 $\pm$.7
ARC-Easy	50.7 $\pm$ 1.0	42.0 $\pm$.7	46.3 $\pm$.2	47.2 $\pm$.9	47.6 $\pm$.6
ARC-Challenge	25.2 $\pm$ 1.1	23.2 $\pm$.4	24.0 $\pm$.5	24.8 $\pm$ 1.1	24.0 $\pm$.5
SVAMP	8.0 $\pm$.9	6.5 $\pm$.7	7.0 $\pm$.6	6.2 $\pm$ 2.2	6.7 $\pm$ 1.5
COPA	61.0 $\pm$ 2.8	57.5 $\pm$.7	60.7 $\pm$.6	61.0 $\pm$.0	63.0 $\pm$ 2.6
PIQA	65.8 $\pm$.2	62.3 $\pm$.1	64.9 $\pm$ 1.3	64.3 $\pm$.2	64.8 $\pm$.3
HellaSwag	36.5 $\pm$.2	29.7 $\pm$.2	33.4 $\pm$.3	34.2 $\pm$.6	34.6 $\pm$.4
WinoGrande	50.7 $\pm$.1	48.6 $\pm$.4	50.1 $\pm$.3	50.2 $\pm$.6	49.8 $\pm$ 1.4
LAMBADA	31.7 $\pm$.9	20.4 $\pm$.4	28.0 $\pm$.4	29.2 $\pm$.9	30.2 $\pm$.9
BoolQ	50.7 $\pm$ 1.6	51.3 $\pm$ 5.0	50.4 $\pm$ 7.7	49.2 $\pm$ 2.7	54.5 $\pm$ 2.4
LSAT-RC	25.7 $\pm$ 1.6	25.2 $\pm$.3	23.4 $\pm$.9	24.1 $\pm$ 1.5	26.4 $\pm$ 1.1
LSAT-LR	26.5 $\pm$ 3.6	27.8 $\pm$ 1.1	28.0 $\pm$ 2.1	25.2 $\pm$ 1.4	27.6 $\pm$ 2.2
SAT-EN	24.8 $\pm$ 3.4	25.0 $\pm$.3	26.2 $\pm$ 2.6	26.5 $\pm$.7	26.5 $\pm$ 1.2
Average	37.1 $\pm$.3	34.2 $\pm$.1	35.9 $\pm$.6	35.9 $\pm$.3	36.9 $\pm$.2

Table 17: **Per-task results at the 119M–285M scale, Tier 5 (Max, about 285M parameters).** Sub-network width 640 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 3840 at $D_{\text{model}} = 512$ for MatFormer, and 1024 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 5 · Max	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	287.5	287.5	282.4	287.5	279.6
Train loss ↓	3.0203 $\pm$.0017	3.3505 $\pm$.0321	3.1436 $\pm$.0035	3.1196 $\pm$.0070	3.0713 $\pm$.0014
Val. loss ↓	3.1908 $\pm$.0014	3.5239 $\pm$.0340	3.3139 $\pm$.0036	3.2899 $\pm$.0071	3.2403 $\pm$.0016
<i>5-shot accuracy (%) ↑</i>					
MMLU	24.8 $\pm$.9	24.5 $\pm$.2	24.8 $\pm$.4	24.3 $\pm$.2	24.2 $\pm$.4
ARC-Easy	50.5 $\pm$.5	41.1 $\pm$.1	46.2 $\pm$.5	47.1 $\pm$.9	48.0 $\pm$.7
ARC-Challenge	25.4 $\pm$.4	23.2 $\pm$ 1.1	24.2 $\pm$.4	25.1 $\pm$ 1.2	24.3 $\pm$.8
SVAMP	5.8 $\pm$.7	6.3 $\pm$ 1.4	7.2 $\pm$ 1.0	5.7 $\pm$ 2.4	6.4 $\pm$ 1.2
COPA	61.0 $\pm$.0	58.0 $\pm$ 2.8	60.0 $\pm$ 1.0	60.7 $\pm$ 1.2	63.3 $\pm$ 2.9
PIQA	66.4 $\pm$.6	61.5 $\pm$.2	64.8 $\pm$ 1.3	64.0 $\pm$.1	65.2 $\pm$.5
HellaSwag	37.6 $\pm$.1	29.9 $\pm$.8	33.4 $\pm$.1	34.2 $\pm$.7	35.1 $\pm$.3
WinoGrande	50.7 $\pm$.1	49.8 $\pm$ 1.7	50.4 $\pm$.9	49.9 $\pm$ 1.1	49.8 $\pm$ 1.4
LAMBADA	33.0 $\pm$ 1.9	20.0 $\pm$.4	27.8 $\pm$.4	29.3 $\pm$ 1.3	30.4 $\pm$.4
BoolQ	49.7 $\pm$ 4.2	55.8 $\pm$ 5.4	50.7 $\pm$ 7.3	50.6 $\pm$ 3.4	52.3 $\pm$ 3.1
LSAT-RC	25.6 $\pm$.3	23.9 $\pm$.5	25.1 $\pm$.6	25.0 $\pm$.6	25.6 $\pm$ 1.1
LSAT-LR	25.7 $\pm$ 2.2	27.2 $\pm$.1	27.0 $\pm$ 3.0	25.4 $\pm$ 2.3	27.6 $\pm$.6
SAT-EN	26.2 $\pm$ 1.4	26.5 $\pm$ 2.4	25.2 $\pm$ 1.0	26.9 $\pm$.7	26.9 $\pm$ 2.0
Average	37.1 $\pm$.7	34.4 $\pm$.5	35.9 $\pm$.7	36.0 $\pm$.5	36.9 $\pm$.5

Table 18: **Per-task results at the 375M–906M scale, Tier 1 (Low, about 375M parameters).** Sub-network width 832 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 1024 at $D_{\text{model}} = 1024$ for MatFormer, and 1024 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 1 · Low	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	377.5	377.5	371.9	377.5	371.9
Train loss ↓	2.8302 $\pm$.0016	3.0381 $\pm$.0070	2.8993 $\pm$.0014	2.8791 $\pm$.0019	2.8746 $\pm$.0035
Val. loss ↓	3.0069 $\pm$.0016	3.2085 $\pm$.0079	3.0724 $\pm$.0015	3.0519 $\pm$.0023	3.0546 $\pm$.0039
<i>5-shot accuracy (%) ↑</i>					
MMLU	24.9 $\pm$.5	25.1 $\pm$.4	25.1 $\pm$.2	23.9 $\pm$.7	24.8 $\pm$.0
ARC-Easy	57.3 $\pm$.8	50.3 $\pm$.5	54.9 $\pm$.6	56.1 $\pm$.4	55.8 $\pm$.8
ARC-Challenge	29.2 $\pm$.3	26.1 $\pm$.2	28.6 $\pm$.8	27.5 $\pm$.2	28.4 $\pm$ 1.4
SVAMP	6.0 $\pm$ 1.5	7.3 $\pm$.3	5.6 $\pm$.4	6.4 $\pm$ 1.3	6.6 $\pm$.2
COPA	67.0 $\pm$.0	62.3 $\pm$ 4.0	67.7 $\pm$ 1.2	67.0 $\pm$ 1.7	67.3 $\pm$ 2.5
PIQA	69.4 $\pm$.5	66.2 $\pm$.6	68.2 $\pm$.5	68.7 $\pm$.3	69.0 $\pm$.7
HellaSwag	46.5 $\pm$.3	37.3 $\pm$.2	43.1 $\pm$.2	44.7 $\pm$.1	45.5 $\pm$.2
WinoGrande	54.0 $\pm$ 1.8	51.8 $\pm$.2	52.7 $\pm$ 1.1	53.9 $\pm$.7	53.9 $\pm$ 1.5
LAMBADA	43.5 $\pm$.6	33.7 $\pm$.8	41.4 $\pm$.7	42.4 $\pm$.9	43.4 $\pm$ 1.5
BoolQ	50.8 $\pm$ 1.0	54.1 $\pm$.6	52.5 $\pm$ 4.8	58.7 $\pm$.3	57.3 $\pm$ 2.4
LSAT-RC	26.1 $\pm$ 2.7	23.1 $\pm$ 1.1	24.0 $\pm$ 2.4	21.5 $\pm$ 3.2	25.0 $\pm$ 3.7
LSAT-LR	23.8 $\pm$ 3.4	28.0 $\pm$.9	26.3 $\pm$ 1.5	25.3 $\pm$ 1.7	27.3 $\pm$ 2.1
SAT-EN	26.9 $\pm$ 2.0	24.3 $\pm$ 2.1	26.7 $\pm$ 3.4	28.0 $\pm$ 1.7	26.1 $\pm$ 3.5
Average	40.4 $\pm$.4	37.7 $\pm$.1	39.7 $\pm$.9	40.3 $\pm$.3	40.8 $\pm$.7

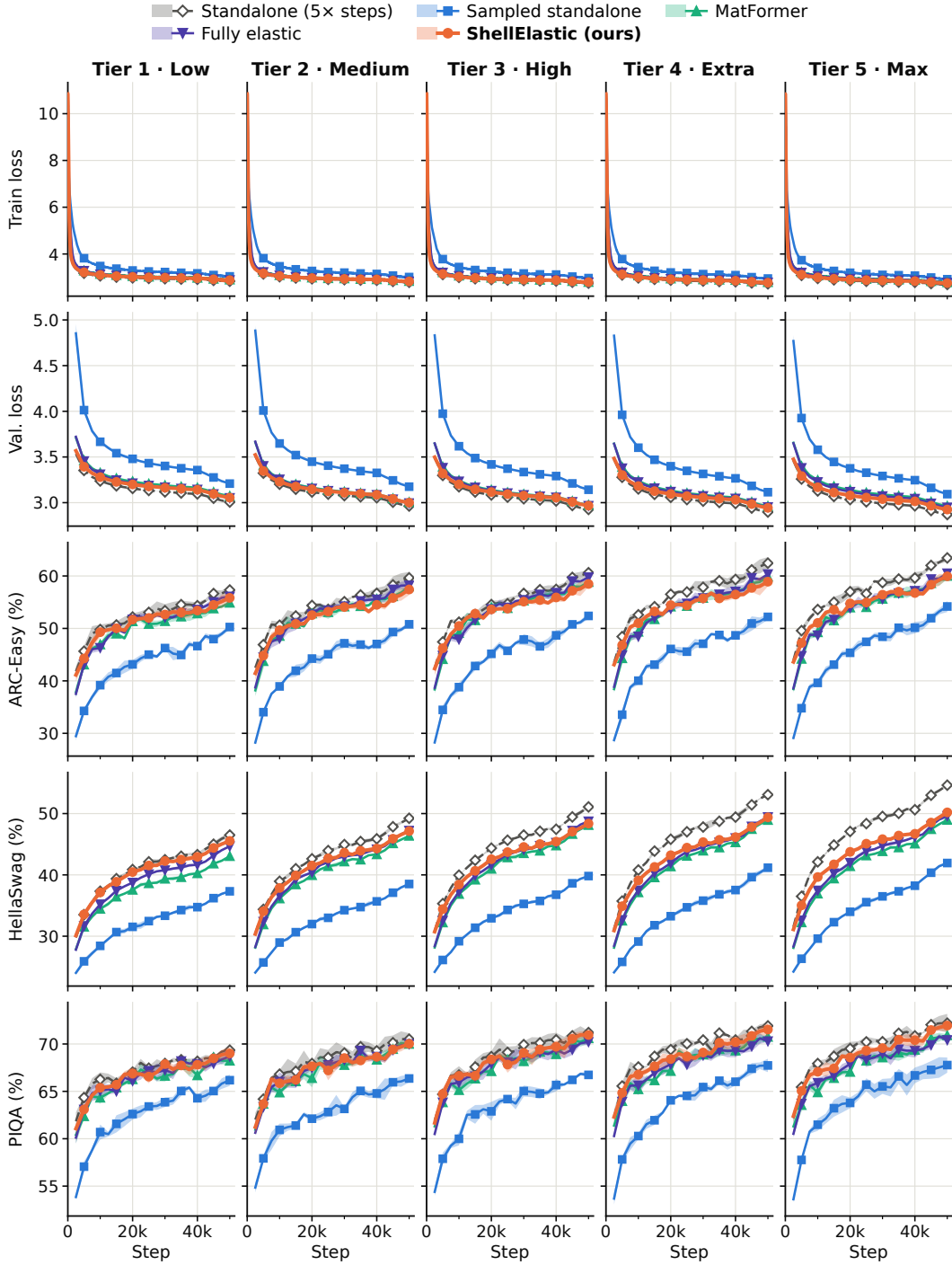


Figure 10: **Training curves at the 375M–906M scale** (Table 2). Columns are the five tiers, from Tier 1 (Low, about 375M parameters) to Tier 5 (Max, about 906M); rows are the train loss, the validation loss and 5-shot accuracy on ARC-Easy, HellaSwag and PIQA. Lines are means over three runs, with shaded bands of ± 1 standard deviation; conventions in Appendix G.

Table 19: **Per-task results at the 375M–906M scale, Tier 2 (Medium, about 486M parameters).** Sub-network width 960 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 2176 at $D_{\text{model}} = 1024$ for MatFormer, and 1280 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 2 · Medium	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	487.7	487.7	485.1	487.7	481.6
Train loss ↓	2.7862 $\pm$.0010	3.0000 $\pm$.0032	2.8283 $\pm$.0015	2.8172 $\pm$.0013	2.8222 $\pm$.0011
Val. loss ↓	2.9632 $\pm$.0010	3.1750 $\pm$.0039	3.0050 $\pm$.0014	2.9948 $\pm$.0015	2.9998 $\pm$.0012
<i>5-shot accuracy (%) ↑</i>					
MMLU	25.2 $\pm$.5	25.2 $\pm$.7	24.7 $\pm$.3	25.2 $\pm$.1	24.8 $\pm$.5
ARC-Easy	59.6 $\pm$ 1.1	50.8 $\pm$.4	58.0 $\pm$.2	58.2 $\pm$.9	57.4 $\pm$ 1.1
ARC-Challenge	30.5 $\pm$.7	26.5 $\pm$ 1.0	29.4 $\pm$.1	29.4 $\pm$.3	28.8 $\pm$.7
SVAMP	6.7 $\pm$.6	6.4 $\pm$ 1.5	6.2 $\pm$ 1.3	6.0 $\pm$ 1.0	5.4 $\pm$ 1.1
COPA	66.3 $\pm$ 2.1	62.0 $\pm$ 1.0	65.0 $\pm$.0	68.3 $\pm$.6	68.0 $\pm$ 3.6
PIQA	70.5 $\pm$.6	66.4 $\pm$.4	70.0 $\pm$.3	70.0 $\pm$.3	70.0 $\pm$.3
HellaSwag	49.2 $\pm$.1	38.5 $\pm$.3	46.4 $\pm$.2	47.2 $\pm$.2	47.1 $\pm$.3
WinoGrande	55.7 $\pm$.5	51.0 $\pm$ 1.8	53.2 $\pm$.8	54.4 $\pm$ 1.0	54.6 $\pm$ 1.5
LAMBADA	46.1 $\pm$ 1.1	34.7 $\pm$.1	44.1 $\pm$.1	45.1 $\pm$ 1.1	44.9 $\pm$ 1.4
BoolQ	54.4 $\pm$.1	52.9 $\pm$ 1.0	53.4 $\pm$ 4.4	55.8 $\pm$ 1.1	57.0 $\pm$.7
LSAT-RC	27.7 $\pm$ 1.8	25.2 $\pm$ 3.2	24.3 $\pm$ 2.6	21.5 $\pm$ 2.4	26.4 $\pm$ 2.8
LSAT-LR	26.9 $\pm$ 2.0	25.2 $\pm$ 2.0	24.4 $\pm$ 2.6	24.3 $\pm$.9	26.7 $\pm$ 2.8
SAT-EN	24.9 $\pm$ 1.0	25.9 $\pm$ 1.0	24.6 $\pm$.3	27.2 $\pm$ 2.6	26.5 $\pm$ 2.2
Average	41.8 $\pm$.4	37.8 $\pm$.3	40.3 $\pm$.3	41.0 $\pm$.3	41.4 $\pm$.7

Table 20: **Per-task results at the 375M–906M scale, Tier 3 (High, about 611M parameters).** Sub-network width 1088 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 3456 at $D_{\text{model}} = 1024$ for MatFormer, and 1536 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 3 · High	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	611.8	611.8	610.9	611.8	608.1
Train loss ↓	2.7496 $\pm$.0018	2.9719 $\pm$.0037	2.8035 $\pm$.0016	2.7933 $\pm$.0011	2.7901 $\pm$.0013
Val. loss ↓	2.9262 $\pm$.0017	3.1414 $\pm$.0040	2.9750 $\pm$.0015	2.9662 $\pm$.0016	2.9670 $\pm$.0016
<i>5-shot accuracy (%) ↑</i>					
MMLU	24.1 $\pm$.5	24.6 $\pm$.9	24.9 $\pm$.9	25.0 $\pm$.8	24.8 $\pm$.3
ARC-Easy	60.6 $\pm$.5	52.4 $\pm$.6	58.8 $\pm$.3	59.8 $\pm$.4	58.5 $\pm$.6
ARC-Challenge	30.6 $\pm$.6	26.7 $\pm$.5	30.0 $\pm$.2	30.3 $\pm$.9	29.1 $\pm$.3
SVAMP	5.0 $\pm$.9	6.0 $\pm$.6	5.9 $\pm$.7	5.9 $\pm$.8	6.6 $\pm$ 1.3
COPA	68.3 $\pm$ 1.5	61.3 $\pm$ 4.0	66.3 $\pm$.6	68.0 $\pm$ 1.7	68.3 $\pm$ 2.5
PIQA	71.2 $\pm$.2	66.7 $\pm$.3	70.6 $\pm$.6	70.2 $\pm$.4	70.9 $\pm$.4
HellaSwag	51.1 $\pm$.2	39.8 $\pm$.1	48.1 $\pm$.2	48.7 $\pm$.2	48.4 $\pm$.1
WinoGrande	55.8 $\pm$ 1.3	52.6 $\pm$.7	53.1 $\pm$.4	54.0 $\pm$ 1.6	54.4 $\pm$ 1.2
LAMBADA	48.3 $\pm$.4	36.0 $\pm$ 1.2	45.2 $\pm$.4	46.8 $\pm$ 1.2	45.5 $\pm$.9
BoolQ	56.5 $\pm$ 4.4	53.0 $\pm$ 2.2	52.1 $\pm$ 4.9	55.6 $\pm$ 3.3	58.2 $\pm$ 1.9
LSAT-RC	28.0 $\pm$ 2.3	24.8 $\pm$ 3.8	23.8 $\pm$ 1.5	24.4 $\pm$ 2.9	26.6 $\pm$.8
LSAT-LR	25.1 $\pm$ 1.2	24.6 $\pm$ 2.3	24.2 $\pm$ 2.2	25.0 $\pm$ 1.5	26.3 $\pm$ 2.2
SAT-EN	27.8 $\pm$ 2.9	25.2 $\pm$ 1.3	22.3 $\pm$ 1.0	27.7 $\pm$.5	28.2 $\pm$ 2.2
Average	42.5 $\pm$.8	38.0 $\pm$.4	40.4 $\pm$.1	41.6 $\pm$.4	42.0 $\pm$.4

Table 21: **Per-task results at the 375M–906M scale, Tier 4 (Extra, about 750M parameters).** Sub-network width 1216 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 4864 at $D_{\text{model}} = 1024$ for MatFormer, and 1792 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 4 · Extra	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	749.8	749.8	749.3	749.8	751.4
Train loss ↓	2.7201 $\pm$.0016	2.9403 $\pm$.0057	2.7857 $\pm$.0015	2.7735 $\pm$.0011	2.7671 $\pm$.0010
Val. loss ↓	2.8972 $\pm$.0016	3.1146 $\pm$.0057	2.9628 $\pm$.0016	2.9520 $\pm$.0014	2.9435 $\pm$.0016
<i>5-shot accuracy (%) ↑</i>					
MMLU	25.1 $\pm$.5	23.9 $\pm$.1	24.6 $\pm$.5	24.4 $\pm$.5	24.5 $\pm$.3
ARC-Easy	62.4 $\pm$ 1.1	52.2 $\pm$.6	59.9 $\pm$.4	60.4 $\pm$.1	58.9 $\pm$.9
ARC-Challenge	32.1 $\pm$.4	26.2 $\pm$.3	30.5 $\pm$.5	30.5 $\pm$ 1.1	29.6 $\pm$.3
SVAMP	5.0 $\pm$.7	6.3 $\pm$ 1.5	6.0 $\pm$ 1.5	6.2 $\pm$.8	5.8 $\pm$.8
COPA	68.7 $\pm$ 1.2	64.7 $\pm$ 3.8	66.0 $\pm$.0	68.3 $\pm$ 1.2	70.0 $\pm$ 4.0
PIQA	71.9 $\pm$.3	67.7 $\pm$.7	70.7 $\pm$.1	70.3 $\pm$.1	71.5 $\pm$.7
HellaSwag	53.1 $\pm$.3	41.2 $\pm$.3	48.9 $\pm$.1	49.4 $\pm$.4	49.4 $\pm$.1
WinoGrande	58.0 $\pm$ 2.0	52.8 $\pm$.7	54.1 $\pm$.8	54.2 $\pm$ 1.6	55.0 $\pm$.8
LAMBADA	49.5 $\pm$.5	39.5 $\pm$.3	45.7 $\pm$.6	47.0 $\pm$ 1.2	46.2 $\pm$ 1.1
BoolQ	59.0 $\pm$ 2.4	53.7 $\pm$ 2.1	54.1 $\pm$ 4.4	56.8 $\pm$ 5.5	57.9 $\pm$.4
LSAT-RC	25.2 $\pm$ 1.8	22.9 $\pm$ 2.2	24.0 $\pm$ 1.7	23.5 $\pm$ 2.4	27.6 $\pm$ 1.6
LSAT-LR	24.5 $\pm$ 1.1	25.8 $\pm$ 1.2	24.7 $\pm$ 1.3	25.6 $\pm$ 1.7	28.0 $\pm$ 1.8
SAT-EN	27.7 $\pm$.0	26.2 $\pm$ 3.9	26.2 $\pm$ 1.9	29.1 $\pm$ 1.0	29.4 $\pm$ 1.4
Average	43.2 $\pm$.5	38.7 $\pm$.5	41.2 $\pm$.3	42.0 $\pm$.5	42.6 $\pm$.3

Table 22: **Per-task results at the 375M–906M scale, Tier 5 (Max, about 906M parameters).** Sub-network width 1344 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 6528 at $D_{\text{model}} = 1024$ for MatFormer, and 2048 for **SHELLELASTIC**. Mean over three runs, with the standard deviation as a subscript.

Tier 5 · Max	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	901.6	901.6	912.9	901.6	911.5
Train loss ↓	2.6922 $\pm$.0012	2.9196 $\pm$.0026	2.7871 $\pm$.0015	2.7710 $\pm$.0009	2.7488 $\pm$.0008
Val. loss ↓	2.8696 $\pm$.0010	3.0931 $\pm$.0032	2.9631 $\pm$.0016	2.9481 $\pm$.0015	2.9250 $\pm$.0012
<i>5-shot accuracy (%) ↑</i>					
MMLU	24.9 $\pm$.5	24.8 $\pm$.2	24.8 $\pm$.0	24.9 $\pm$.5	24.8 $\pm$.1
ARC-Easy	63.4 $\pm$.8	54.2 $\pm$ 1.1	60.0 $\pm$.7	60.5 $\pm$.1	59.9 $\pm$.4
ARC-Challenge	33.0 $\pm$.6	26.8 $\pm$.8	30.6 $\pm$.6	30.4 $\pm$.7	31.0 $\pm$.8
SVAMP	5.6 $\pm$.7	4.9 $\pm$.8	5.8 $\pm$.8	6.4 $\pm$.5	6.0 $\pm$ 1.3
COPA	70.0 $\pm$ 2.0	61.7 $\pm$ 2.1	66.3 $\pm$.6	67.7 $\pm$ 1.2	72.0 $\pm$ 2.6
PIQA	72.2 $\pm$ 1.0	67.8 $\pm$.8	70.9 $\pm$.1	70.4 $\pm$.3	71.9 $\pm$.6
HellaSwag	54.6 $\pm$.2	41.9 $\pm$.1	49.0 $\pm$.1	49.6 $\pm$.5	50.2 $\pm$.2
WinoGrande	57.6 $\pm$ 1.4	53.2 $\pm$.7	54.2 $\pm$ 1.0	54.3 $\pm$.9	54.9 $\pm$.9
LAMBADA	51.4 $\pm$.8	39.0 $\pm$.4	45.4 $\pm$.8	47.0 $\pm$.8	46.3 $\pm$ 1.2
BoolQ	54.8 $\pm$ 5.4	57.2 $\pm$ 1.7	55.7 $\pm$ 4.1	57.9 $\pm$ 4.6	59.8 $\pm$ 1.7
LSAT-RC	24.5 $\pm$ 2.6	24.6 $\pm$.4	24.6 $\pm$ 1.0	24.1 $\pm$.2	27.4 $\pm$ 2.1
LSAT-LR	24.1 $\pm$.7	26.3 $\pm$ 3.2	25.0 $\pm$ 1.6	25.2 $\pm$.3	27.1 $\pm$.5
SAT-EN	24.8 $\pm$ 3.8	24.8 $\pm$ 1.8	24.4 $\pm$ 1.2	28.0 $\pm$.6	27.5 $\pm$ 1.7
Average	43.1 $\pm$.8	39.0 $\pm$.5	41.3 $\pm$.6	42.0 $\pm$.4	43.0 $\pm$.3

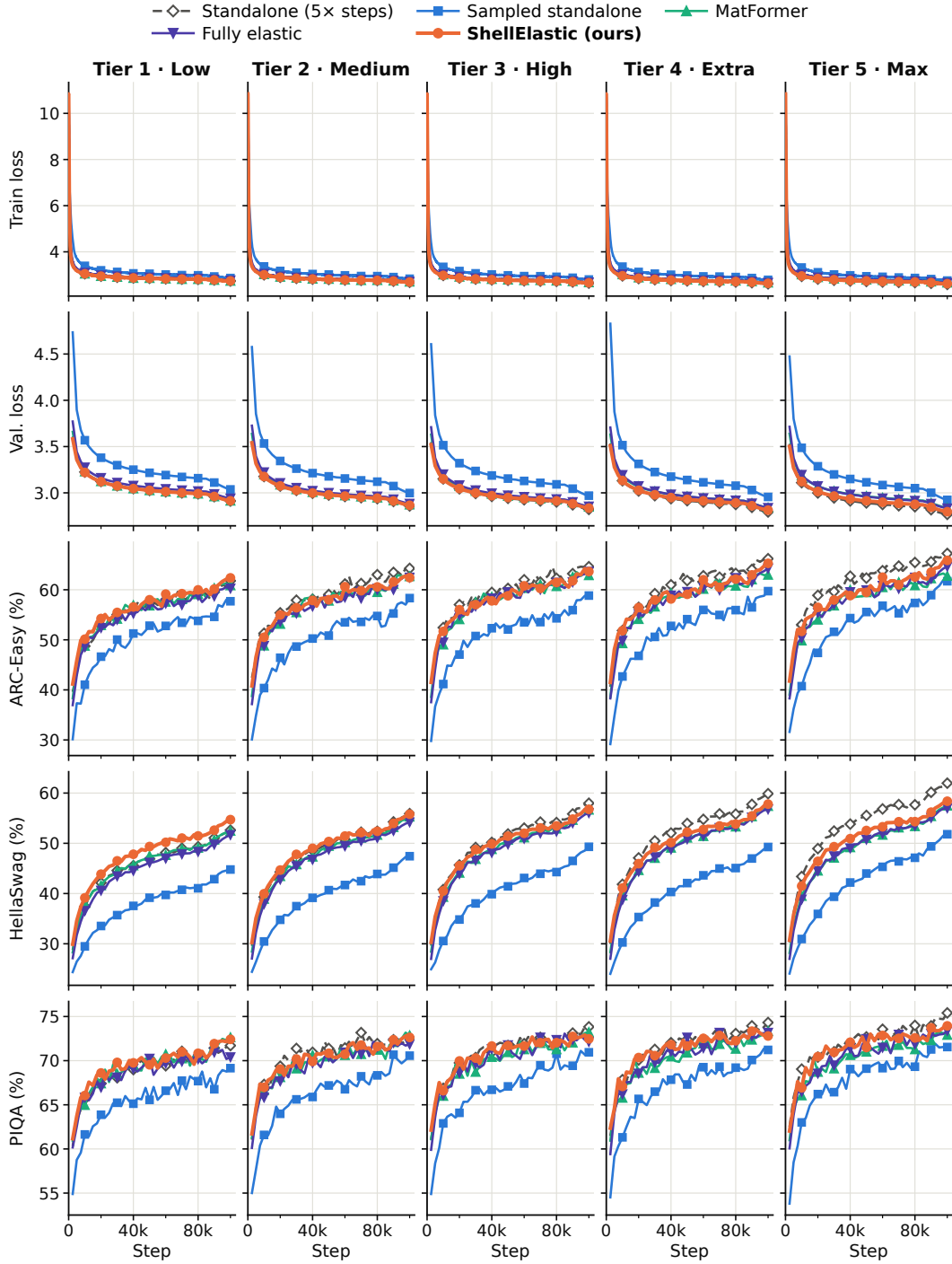


Figure 11: **Training curves at the 675M–1.69B scale** (Table 2). Columns are the five tiers, from Tier 1 (Low, about 675M parameters) to Tier 5 (Max, about 1.69B); rows are the train loss, the validation loss and 5-shot accuracy on ARC-Easy, HellaSwag and PIQA. Each line is a single run; conventions in Appendix G.

Table 23: **Per-task results at the 675M–1.69B scale, Tier 1 (Low, about 675M parameters).** Sub-network width 832 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 4096 at $D_{\text{model}} = 1024$ for MatFormer, and 1024 for **SHELLELASTIC**. Single runs.

Tier 1 · Low	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	676.6	676.6	673.8	676.6	673.8
Train loss ↓	2.7335	2.8643	2.7365	2.7638	2.7347
Val. loss ↓	2.9118	3.0370	2.9140	2.9403	2.9135
<i>5-shot accuracy (%) ↑</i>					
MMLU	25.2	23.6	24.1	23.3	24.1
ARC-Easy	61.8	57.7	60.9	60.3	62.4
ARC-Challenge	31.4	29.6	32.3	30.2	32.2
SVAMP	6.3	6.3	6.7	4.7	4.7
COPA	70.0	64.0	67.0	71.0	70.0
PIQA	71.7	69.2	72.6	70.4	72.4
HellaSwag	52.6	44.8	52.5	51.7	54.7
WinoGrande	57.1	50.8	56.7	56.9	58.2
LAMBADA	47.1	43.0	49.7	44.0	51.6
BoolQ	57.3	52.4	60.6	55.7	61.9
LSAT-RC	29.1	29.5	23.9	26.9	26.1
LSAT-LR	24.9	28.4	27.5	29.0	24.9
SAT-EN	29.1	23.8	22.3	22.8	25.2
Average	43.4	40.2	42.8	42.1	43.7

Table 24: **Per-task results at the 675M–1.69B scale, Tier 2 (Medium, about 882M parameters).** Sub-network width 960 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 6144 at $D_{\text{model}} = 1024$ for MatFormer, and 1280 for **SHELLELASTIC**. Single runs.

Tier 2 · Medium	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	885.8	885.8	875.2	885.8	878.0
Train loss ↓	2.6791	2.8253	2.6938	2.7067	2.6865
Val. loss ↓	2.8587	2.9980	2.8720	2.8840	2.8657
<i>5-shot accuracy (%) ↑</i>					
MMLU	23.8	24.0	24.6	24.1	23.6
ARC-Easy	64.3	58.3	62.7	62.5	62.4
ARC-Challenge	33.8	28.2	33.4	31.6	32.9
SVAMP	6.0	7.0	7.0	5.0	6.0
COPA	73.0	63.0	69.0	73.0	65.0
PIQA	72.6	70.6	72.9	71.9	72.6
HellaSwag	55.9	47.4	55.3	54.2	55.8
WinoGrande	59.0	52.7	57.5	58.2	58.8
LAMBADA	52.0	43.8	51.2	48.2	52.4
BoolQ	55.0	59.0	62.6	55.8	62.8
LSAT-RC	26.1	23.1	20.9	27.6	23.5
LSAT-LR	25.3	27.8	25.5	27.5	27.3
SAT-EN	23.8	24.8	20.9	24.8	24.8
Average	43.9	40.8	43.3	43.4	43.7

Table 25: **Per-task results at the 675M–1.69B scale, Tier 3 (High, about 1.12B parameters).** Sub-network width 1088 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 8704 at $D_{\text{model}} = 1024$ for MatFormer, and 1536 for **SHELLELASTIC**. Single runs.

Tier 3 · High	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	1123.2	1123.2	1126.8	1123.2	1117.7
Train loss ↓	2.6424	2.7994	2.6730	2.6788	2.6550
Val. loss ↓	2.8212	2.9696	2.8487	2.8539	2.8351
<i>5-shot accuracy (%) ↑</i>					
MMLU	25.3	22.9	24.3	24.3	24.2
ARC-Easy	64.6	58.8	62.9	63.6	63.6
ARC-Challenge	34.3	30.1	33.4	32.8	33.8
SVAMP	6.3	5.3	7.0	5.0	7.0
COPA	72.0	68.0	69.0	71.0	67.0
PIQA	73.8	70.9	73.2	72.1	72.5
HellaSwag	58.0	49.3	56.7	56.1	56.8
WinoGrande	59.7	54.9	59.4	58.7	58.6
LAMBADA	52.7	45.8	52.7	50.2	52.4
BoolQ	61.8	54.2	61.3	57.9	62.1
LSAT-RC	22.4	26.1	23.5	25.4	22.8
LSAT-LR	24.5	28.0	23.3	28.8	24.7
SAT-EN	25.7	25.7	18.0	25.2	25.7
Average	44.7	41.6	43.4	43.9	43.9

Table 26: **Per-task results at the 675M–1.69B scale, Tier 4 (Extra, about 1.39B parameters).** Sub-network width 1216 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 11520 at $D_{\text{model}} = 1024$ for MatFormer, and 1792 for **SHELLELASTIC**. Single runs.

Tier 4 · Extra	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	1388.5	1388.5	1403.7	1388.5	1393.1
Train loss ↓	2.6122	2.7838	2.6617	2.6621	2.6323
Val. loss ↓	2.7933	2.9557	2.8385	2.8384	2.8131
<i>5-shot accuracy (%) ↑</i>					
MMLU	23.7	24.6	24.2	24.8	24.2
ARC-Easy	66.2	59.7	63.0	64.3	65.3
ARC-Challenge	35.1	30.0	32.8	33.6	32.9
SVAMP	6.7	2.7	8.0	3.7	6.0
COPA	73.0	64.0	72.0	73.0	68.0
PIQA	74.3	71.2	73.3	73.2	72.8
HellaSwag	59.9	49.3	57.4	56.8	57.8
WinoGrande	59.7	54.2	59.3	58.8	59.2
LAMBADA	55.2	46.6	53.3	51.6	52.5
BoolQ	64.6	56.3	61.3	59.2	62.4
LSAT-RC	24.6	25.0	23.1	26.1	24.3
LSAT-LR	28.4	22.7	25.9	29.2	25.1
SAT-EN	29.1	27.2	18.0	20.9	25.7
Average	46.2	41.0	44.0	44.2	44.3

Table 27: **Per-task results at the 675M–1.69B scale, Tier 5 (Max, about 1.69B parameters).** Sub-network width 1344 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 14592 at $D_{\text{model}} = 1024$ for MatFormer, and 2048 for **SHELLELASTIC**. Single runs.

Tier 5 · Max	Standalone [†]	Sampled standalone	MatFormer	Fully elastic	SHELLELASTIC
Params (M)	1682.0	1682.0	1705.6	1682.0	1704.2
Train loss ↓	2.5859	2.7509	2.6585	2.6549	2.6152
Val. loss ↓	2.7674	2.9242	2.8378	2.8334	2.7959
<i>5-shot accuracy (%) ↑</i>					
MMLU	25.2	24.7	24.2	24.3	24.6
ARC-Easy	67.3	61.7	62.8	64.6	65.9
ARC-Challenge	36.6	32.4	32.6	34.1	33.9
SVAMP	5.0	4.7	6.3	5.3	6.0
COPA	73.0	66.0	72.0	72.0	66.0
PIQA	75.4	71.5	72.9	73.4	73.9
HellaSwag	62.0	51.8	57.5	57.0	58.4
WinoGrande	60.5	55.0	59.2	60.2	59.2
LAMBADA	56.9	47.6	53.5	52.2	52.4
BoolQ	63.0	61.4	60.6	59.3	62.5
LSAT-RC	25.0	27.2	20.1	27.2	22.4
LSAT-LR	22.7	28.6	24.1	28.0	27.8
SAT-EN	26.7	29.1	20.4	21.4	25.2
Average	46.1	43.2	43.6	44.6	44.5

H ADDITIONAL TRANSFER RESULTS

Figures 12–14 repeat the sweeps of Figure 3 for each of the five sub-networks of MatFormer, the fully elastic model and **SHELLELASTIC**. Lines, stars and Δ val. loss are as in Figure 3, with the loss of each sub-network in place of the mean over the five; the proxy line is repeated in every column, and the five panels of a row share one y-range. Two observations emerge:

- **μ -ELASTIC transfers to every sub-network.** Under μ -ELASTIC, the optimal η_0 of every sub-network lies between 10^{-3} and 3.3×10^{-3} and the optimal σ_0 at 3.3×10^{-3} or 10^{-2} , and the base values of Section 4.1 are within 0.005 nats of the optimum of every sub-network at the 119M–285M and 375M–906M scales.
- **The gap between SP and μ -ELASTIC is largest at the narrow widths.** The gap is larger at the smaller sub-networks at both scales, for example 0.022 nats at Low and 0.007 at Max for **SHELLELASTIC** in the η_0 sweep at the 119M–285M scale, as in Table 3, and 0.015 at Low and 0.003 at Max at the 375M–906M scale.

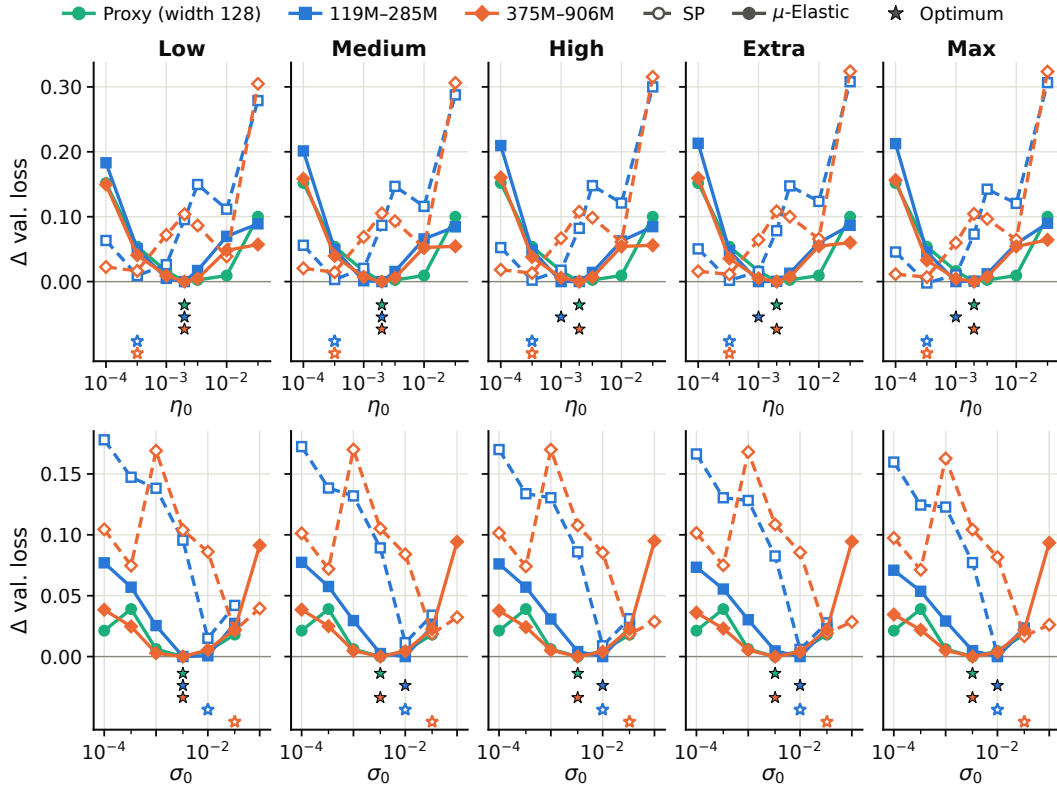


Figure 12: **Per-sub-network sweeps for MatFormer.** The sweeps of Figure 3 for each sub-network, from the smallest (Low) to the largest (Max; widths in Table 7), with the loss of each sub-network in place of the mean over the five.

Weight decay. Figure 15 sweeps the weight decay λ_0 over seven values for the proxy and for **SHELLELASTIC** at the 119M–285M scale, with η_0 and σ_0 at the base values of Section 4.1, and Figure 16 shows each sub-network. Two observations emerge:

- **Weight decay transfers under μ -ELASTIC.** From $\lambda_0 = 10^{-3}$ to 10^{-1} , the mean loss of **SHELLELASTIC** under μ -ELASTIC varies by at most 0.003 nats, the same flat range as the proxy’s, and both rise from 3.3×10^{-1} on; the base value is within 0.001 nats of the best μ -ELASTIC value.
- **μ -ELASTIC is lower than SP at every swept value.** From 3.3×10^{-3} to 10^{-1} , where SP is also flat (within 0.009 nats), the gap is 0.018–0.028 nats in the mean and again largest at

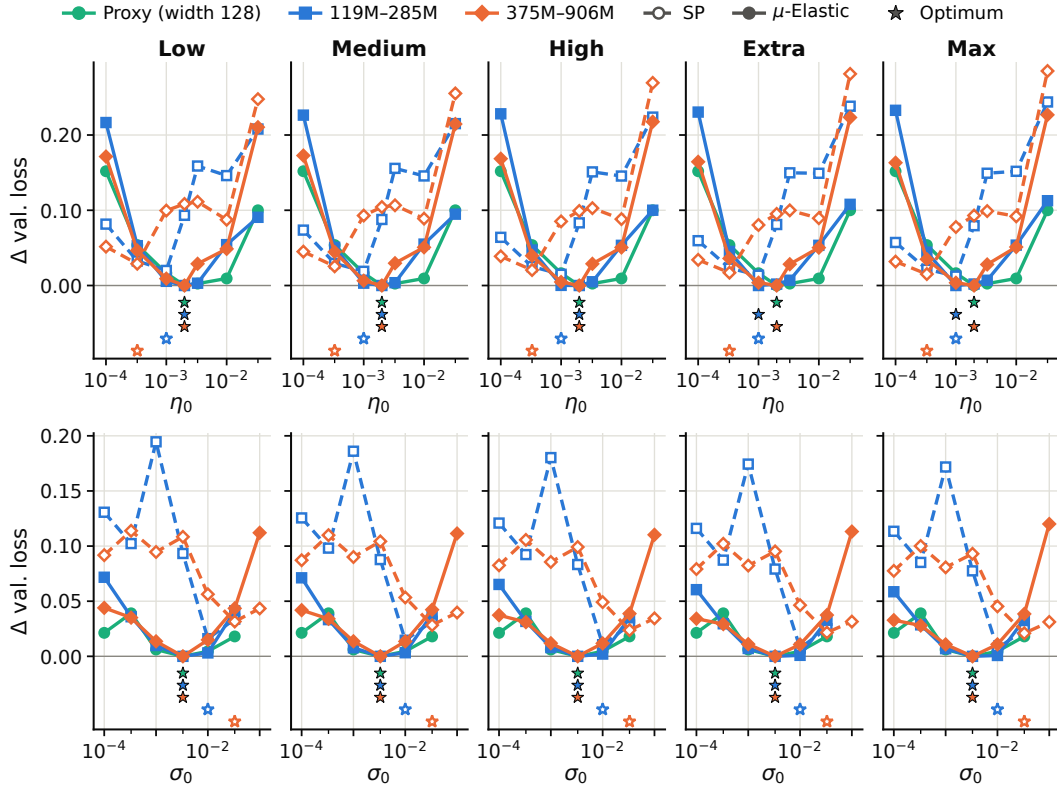


Figure 13: Per-sub-network sweeps for the fully elastic model, as in Figure 12.

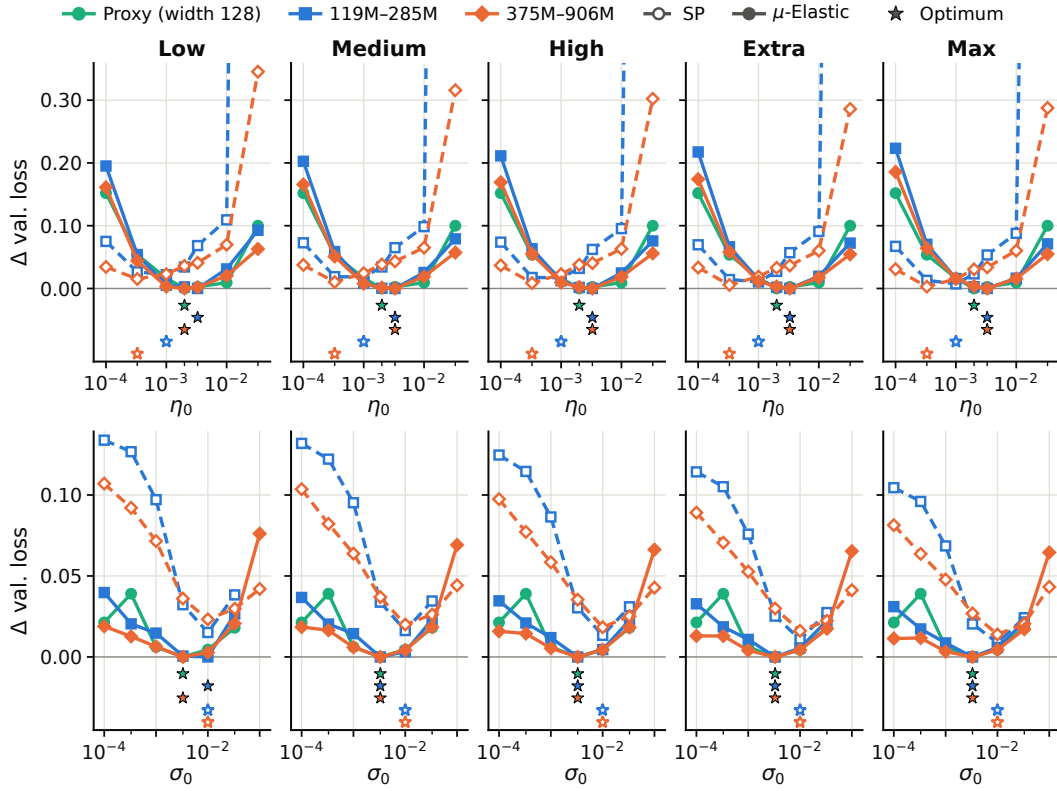


Figure 14: Per-sub-network sweeps for SHELLELASTIC, as in Figure 12.

the narrow widths, 0.023–0.032 nats at Low against 0.010–0.020 at Max. SP also degrades at 10^{-3} , where the gap grows to 0.039 nats, and the gap shrinks to 0.010 and 0.005 nats at 3.3×10^{-1} and 1, where both degrade.

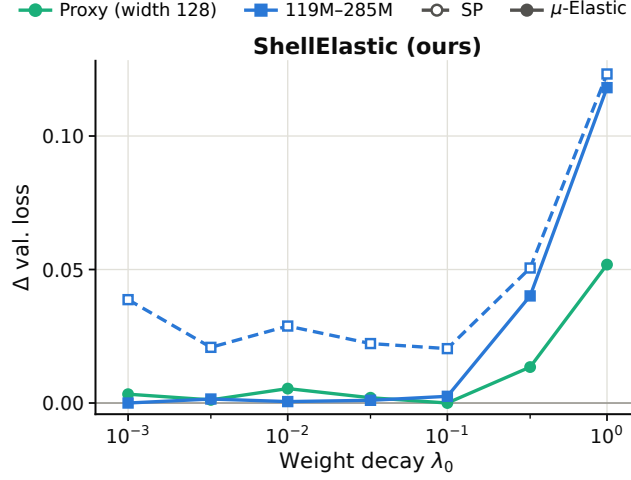


Figure 15: **Weight-decay sweeps.** Sweeps of λ_0 for the proxy under μP and for **SHELLELASTIC** at the 119M–285M scale under μ -ELASTIC (solid) and SP (dashed, open markers); Δ val. loss is as in Figure 3.

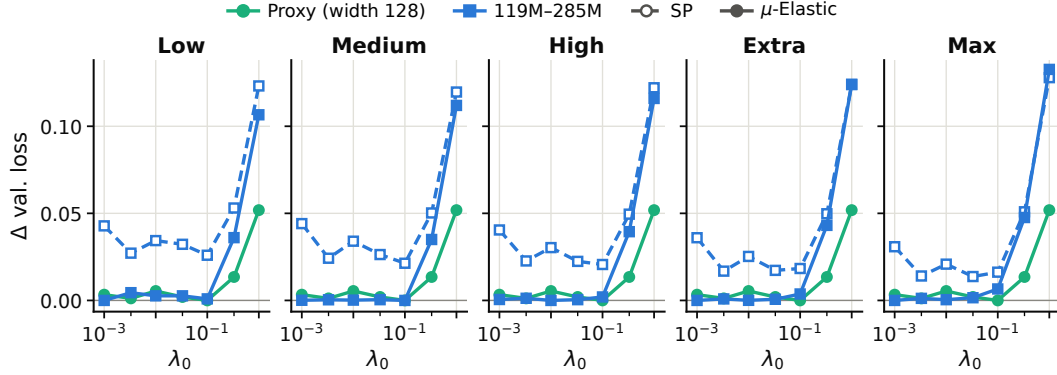


Figure 16: **Per-sub-network weight-decay sweeps.** The sweeps of Figure 15 for each sub-network of **SHELLELASTIC**, from the smallest (Low) to the largest (Max); lines and Δ val. loss as there, with the loss of each sub-network in place of the mean over the five, and one y-range for all panels.

Table 28: **Effect of μ -ELASTIC on each elastic model at the 119M–285M and 375M–906M scales**, the values of Table 3 with four decimals.

Method	Low	Medium	High	Extra	Max	Avg.
119M–285M, train loss ↓						
MatFormer	3.2614	3.1841	3.1566	3.1405	3.1393	3.1764
w/ μ -ELASTIC	3.2536	3.1821	3.1561	3.1407	3.1436	3.1752
Fully elastic	3.2717	3.1917	3.1544	3.1304	3.1263	3.1749
w/ μ -ELASTIC	3.2572	3.1783	3.1447	3.1233	3.1196	3.1646
SHELL ELASTIC	3.2472	3.1801	3.1356	3.1040	3.0794	3.1493
w/ μ -ELASTIC	3.2305	3.1641	3.1229	3.0941	3.0713	3.1366
119M–285M, val. loss ↓						
MatFormer	3.4341	3.3598	3.3271	3.3138	3.3110	3.3492
w/ μ -ELASTIC	3.4248	3.3566	3.3250	3.3123	3.3139	3.3465
Fully elastic	3.4451	3.3681	3.3251	3.3030	3.2969	3.3477
w/ μ -ELASTIC	3.4291	3.3536	3.3145	3.2954	3.2899	3.3365
SHELL ELASTIC	3.4232	3.3538	3.3077	3.2749	3.2491	3.3217
w/ μ -ELASTIC	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090
375M–906M, train loss ↓						
MatFormer	2.9140	2.8413	2.8152	2.7958	2.7929	2.8318
w/ μ -ELASTIC	2.8993	2.8283	2.8035	2.7857	2.7871	2.8208
Fully elastic	2.9061	2.8418	2.8133	2.7900	2.7856	2.8274
w/ μ -ELASTIC	2.8791	2.8172	2.7933	2.7735	2.7710	2.8068
SHELL ELASTIC	2.8877	2.8303	2.7963	2.7696	2.7481	2.8064
w/ μ -ELASTIC	2.8746	2.8222	2.7901	2.7671	2.7488	2.8006
375M–906M, val. loss ↓						
MatFormer	3.0889	3.0194	2.9881	2.9741	2.9700	3.0081
w/ μ -ELASTIC	3.0724	3.0050	2.9750	2.9628	2.9631	2.9957
Fully elastic	3.0808	3.0202	2.9867	2.9690	2.9633	3.0040
w/ μ -ELASTIC	3.0519	2.9948	2.9662	2.9520	2.9481	2.9826
SHELL ELASTIC	3.0697	3.0087	2.9732	2.9461	2.9243	2.9844
w/ μ -ELASTIC	3.0546	2.9998	2.9670	2.9435	2.9250	2.9780

I ARCHITECTURE COMPARISON AND SEARCH

This appendix complements Section 4.3: Appendix I.1 gives the per-task results and training curves of StairFormer, and Appendix I.2 details the architecture search at equal parameters and width count.

I.1 COMPARISON WITH STAIRFORMER

StairFormer (Trost et al., 2026) nests sub-networks as prefixes of one forward pass with equal-width groups, $\mathcal{S} = \{1, \dots, M\}$ in our notation; it normalizes over cumulative prefixes (Equation 25) and shares one terminal-norm gain and one readout across sub-networks. Our StairFormer baseline has five 208-wide groups ($g = 208$, $\mathcal{S} = \{1, 2, 3, 4, 5\}$) with $d_{\text{head}} = 104$, so that each group holds two complete heads, and an FFN hidden width of 236 per group ($e \approx 1.13$, against $e = 1$ for **SHELLELASTIC**), so that its Max matches that of **SHELLELASTIC** in parameters. Like **SHELLELASTIC**, it trains all sub-networks at every step; we give it the data, steps, HPs and loss weights of Section 4.1, and μ -**ELASTIC** with $m = 208/128 = 1.625$ (Appendix D). Table 29 gives its per-task results at every tier, and Figure 17 its training curves next to the methods of Table 2 other than sampled standalone. Two observations emerge:

- **Only Max is parameter-matched.** StairFormer’s Max and that of **SHELLELASTIC** both have 279.6M parameters, but at Tiers 1–4, StairFormer’s sub-networks are 14–73% smaller (32.6–200.4M against 118.8–233.1M), so these pairs compare sub-networks of different sizes; Figure 4 compares the validation loss and average 5-shot accuracy against parameter count.
- **At Max, **SHELLELASTIC** is better on most tasks.** At the matched size, **SHELLELASTIC** is better on 9 of the 13 tasks and worse on 4 (MMLU, SVAMP, WinoGrande and LSAT-RC).

I.2 ARCHITECTURE SEARCH AT EQUAL PARAMETERS AND WIDTH COUNT

The runs of Figures 4c and 19 train every supported width at every step with the default recipe and change only $\mathcal{S}$. A supported set fixes how many of the M^2 blocks of size $g \times g$ a hidden matrix keeps: group $\mathcal{G}_k$ has $s_k - s_{k-1}$ block rows, each reading s_k input blocks, so

$$B(\mathcal{S}) = \sum_{k=1}^K (s_k - s_{k-1}) s_k. \quad (24)$$

We compare two families of supported sets (Figure 18): five supported widths at $B = 42$, the block count of the default $\mathcal{S} = \{4, 5, 6, 7, 8\}$, and six supported widths at $B = 39$. Every set contains exactly one group wider than g , and the sets of a family differ only in where this group sits, from the last shell to the core. A width at or above the wide group has the same number of blocks wherever the group sits, so its parameter count is the same in every set of the family; a width below it keeps only the lower-triangle blocks and has fewer parameters (63.9M at 384, 93.7M at 512, 127.6M at 640; marked [†] in Table 30). The average in Table 30 is taken over the supported widths of each set, so it covers different widths in different rows, and the core placements have no width below the core.

Table 29: **Per-task results of StairFormer and SHELLELASTIC at every tier** of the 119M–285M scale, with the rows of Appendix G. Stair.: StairFormer. Ours: **SHELLELASTIC**. Bold marks the better entry of each pair (ties both bold). Only Tier 5 is parameter-matched.

	Tier 1 · Low		Tier 2 · Medium		Tier 3 · High		Tier 4 · Extra		Tier 5 · Max	
	Stair.	Ours	Stair.	Ours	Stair.	Ours	Stair.	Ours	Stair.	Ours
Params (M)	32.6	118.8	76.9	152.7	132.8	190.8	200.4	233.1	279.6	279.6
Train loss ↓	3.6874	3.2305	3.4224	3.1641	3.2764	3.1229	3.1871	3.0941	3.1258	3.0713
Val. loss ↓	3.8836	3.4086	3.6004	3.3375	3.4519	3.2944	3.3588	3.2641	3.2949	3.2403
<i>5-shot accuracy (%) ↑</i>										
MMLU	25.1	24.7	25.1	23.8	25.1	24.0	25.5	24.4	25.3	24.2
ARC-Easy	35.7	45.0	41.3	45.8	41.8	47.1	44.0	47.6	45.7	48.0
ARC-Challenge	21.5	23.3	22.4	23.5	22.3	23.6	22.5	24.0	23.2	24.3
SVAMP	6.0	5.7	5.3	6.3	7.3	6.3	7.0	6.7	8.3	6.4
COPA	55.0	59.7	57.0	61.7	60.0	61.7	56.0	63.0	59.0	63.3
PIQA	57.8	62.1	60.2	63.7	61.8	64.6	63.1	64.8	64.0	65.2
HellaSwag	27.0	32.4	29.7	33.2	31.1	33.9	32.2	34.6	33.6	35.1
WinoGrande	52.6	50.7	51.9	49.3	51.8	50.0	52.2	49.8	50.4	49.8
LAMBADA	15.0	25.9	23.3	29.1	25.3	29.9	26.8	30.2	27.4	30.4
BoolQ	47.0	55.1	48.4	57.2	53.2	55.3	51.1	54.5	51.7	52.3
LSAT-RC	25.4	24.8	23.1	25.6	22.4	26.0	25.4	26.4	28.0	25.6
LSAT-LR	25.7	26.2	26.5	29.2	24.1	26.8	24.5	27.6	24.1	27.6
SAT-EN	25.7	25.6	27.2	27.2	24.3	26.2	24.3	26.5	23.8	26.9
Average	32.3	35.5	34.0	36.6	34.7	36.6	35.0	36.9	35.7	36.9

Table 30: **Validation loss of every supported width in the architecture search**, at step 25,000, as the one group wider than g moves from the last shell (first row of each block) to the core (last row, bold). Params: non-zero parameters (M) of each width; [†]a width below the wide group, which has fewer parameters. Avg.: mean over the supported widths of each set. Bold marks the lowest value in each column.

$\mathcal{S}$	128	256	384	512	640	768	896	1024	Avg.
<i>B = 42, 5 supported widths</i>									
Params	17.1	38.4	63.9	118.8	152.7	190.8	233.1	279.6	
{1, 2, 3, 4, 8}	4.1178	3.8183	3.6738	3.6023 [†]	–	–	–	3.2698	3.6964
{1, 2, 3, 7, 8}	4.1037	3.8310	3.7224	–	–	–	3.2910	3.2503	3.6397
{1, 2, 6, 7, 8}	4.0895	3.8832	–	–	–	3.3193	3.2721	3.2400	3.5608
{1, 5, 6, 7, 8}	4.1154	–	–	–	3.3657	3.3068	3.2687	3.2409	3.4595
{4, 5, 6, 7, 8}	–	–	–	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090
<i>B = 39, 6 supported widths</i>									
Params	17.1	38.4	76.5	106.2	140.1	178.2	220.5	267.0	
{1, 2, 3, 4, 5, 8}	4.1310	3.8646	3.6689 [†]	3.5601 [†]	3.4983 [†]	–	–	3.3314	3.6757
{1, 2, 3, 4, 7, 8}	4.1371	3.8356	3.6898 [†]	3.6098 [†]	–	–	3.3460	3.3035	3.6537
{1, 2, 3, 6, 7, 8}	4.1112	3.8520	3.7369 [†]	–	–	3.3659	3.3134	3.2775	3.6095
{1, 2, 5, 6, 7, 8}	4.1022	3.8981	–	–	3.4082	3.3456	3.3047	3.2734	3.5554
{1, 4, 5, 6, 7, 8}	4.1499	–	–	3.4746	3.3919	3.3419	3.3057	3.2778	3.4903
{3, 4, 5, 6, 7, 8}	–	–	3.5445	3.4391	3.3818	3.3371	3.3050	3.2798	3.3812

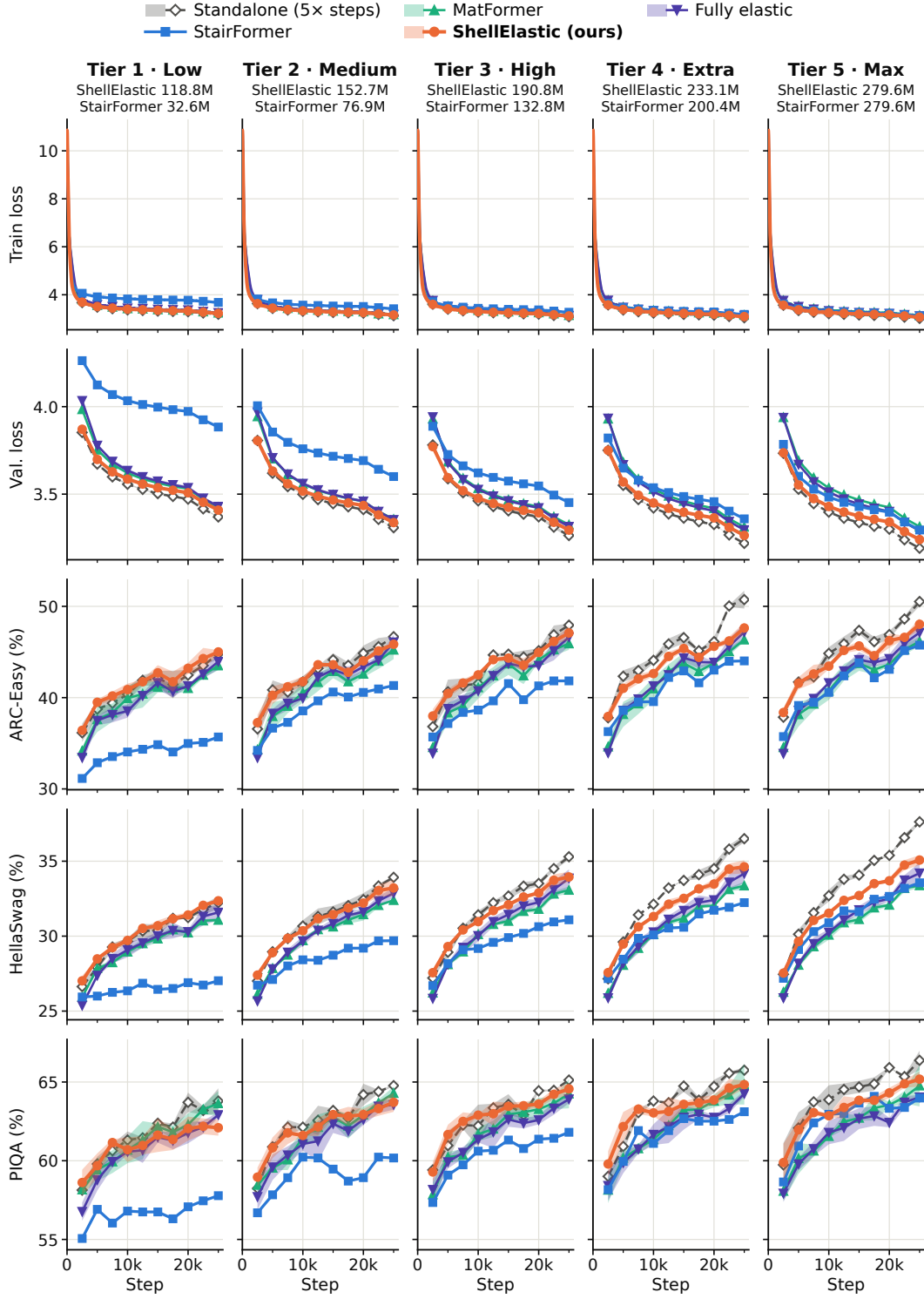


Figure 17: **Training curves of StairFormer and the methods of Table 2** at the 119M–285M scale, except sampled standalone, in the format of Figure 9; column titles give the parameters of SHELLELASTIC and StairFormer.

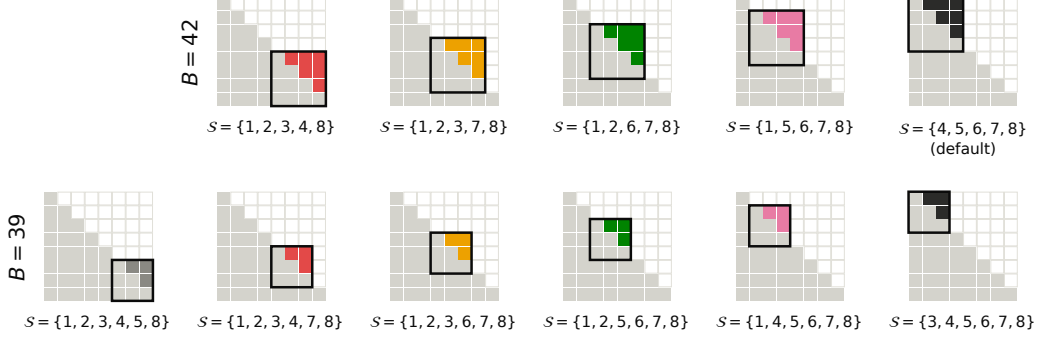


Figure 18: **Block masks of the supported sets in Figures 4c and 19** ($M = 8, g = 128$). Cell (i, j) is the $g \times g$ block of every hidden matrix from input block j to output block i . Gray: lower-triangle blocks kept by every set; colored: blocks above the diagonal, all inside the one group wider than g (outlined). Each row has the same B and moves the wide group from the last shell (left) to the core (right).

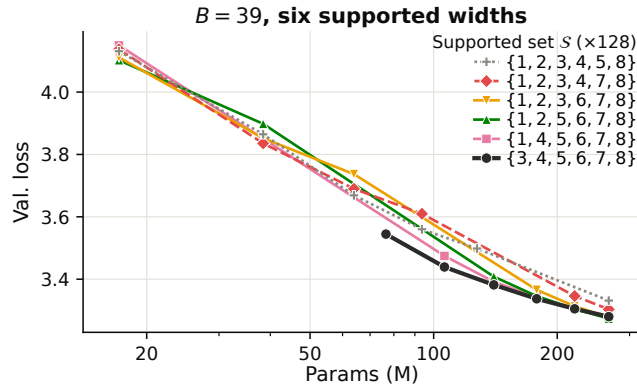


Figure 19: As Figure 4c, at $B = 39$ with six supported widths (core placement $S = \{3, 4, 5, 6, 7, 8\}$, black).

J ABLATION STUDIES

All ablations use the supernet of Section 4.1 ($g = 128$, $\mathcal{S} = \{4, 5, 6, 7, 8\}$) at the 119M–285M scale and change one setting of its recipe at a time. Tables 31 and 32 list all results, numbered as the paragraphs below; each block of rows ends with the default.

- **Default and run-to-run variation.** The default rows average the three **SHELLELASTIC** runs of Table 2, whose standard deviation is at most 0.003 nats in the validation loss of any tier, 0.002 nats in the mean validation loss and 0.32 points in the mean 5-shot accuracy; every other row is a single run. A further run of the default on H100 GPUs is within 0.003 nats of this mean at every tier, so we treat differences below 0.003 nats as run-to-run variation.
- **Hardware and cost.** The default runs used A100 GPUs with activation checkpointing on every other block, as did ablations (11) and (12), whereas ablations (1) to (10) used H100 GPUs without it. Step times and peak memory are compared between runs on the same GPUs with the same activation checkpointing; on H100 GPUs, the reference is the loss-weight runs of (2) (37.1 GB peak memory).

Table 31: **Ablations (1)–(6) of SHELLELASTIC at the 119M–285M scale.** Each block changes one setting of the default recipe and ends with the default. Val. loss is taken at step 25,000 for each tier (Low to Max: widths 512 to 1024), and 5-shot acc. averages the 13 downstream tasks over the five tiers. Bold marks the best value of each column within a block.

Setting	Val. loss ↓						5-shot acc. ↑
	Low	Medium	High	Extra	Max	Avg.	Avg.
<i>(1) Selected sub-networks per step $\mathcal{A}_\tau$</i>							
Sandwich: Low, Max and one other	3.4019	3.3617	3.3183	3.2816	3.2395	3.3206	36.1
Random: one of the five sub-networks	3.4311	3.3786	3.3542	3.3410	3.3356	3.3681	34.8
Max + random: Max and one other	3.4944	3.4079	3.3447	3.2956	3.2397	3.3564	35.5
All: all five sub-networks (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(2) Loss weights ω_s (Low to Max)</i>							
(1, 1, 1, 1, 1)	3.3888	3.3279	3.2923	3.2677	3.2518	3.3057	36.5
(1, 0.5, 0.3, 0.2, 2)	3.3801	3.3312	3.2978	3.2690	3.2412	3.3039	36.6
(1, 1, 1, 1, 2)	3.3959	3.3317	3.2934	3.2673	3.2486	3.3074	36.3
(1, 1, 1, 1, 6)	3.4236	3.3437	3.2977	3.2638	3.2365	3.3130	36.2
(1, 1, 1, 1, 4) (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(3) Trunk normalization</i>							
Cumulative prefix	3.4060	3.3376	3.2952	3.2645	3.2410	3.3088	36.1
Block-local over every g coordinates	3.4093	3.3368	3.2955	3.2652	3.2411	3.3096	36.7
Group-local over $\mathcal{G}_k$ (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(4) Terminal norm N_s^{out}</i>							
Group-local, as the trunk	3.4122	3.3414	3.2975	3.2656	3.2417	3.3117	35.4
Over all sg coordinates (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(5) Readout</i>							
Shared U	3.4152	3.3446	3.3000	3.2696	3.2450	3.3149	35.6
Separate U_s (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(6) In-place distillation from Max</i>							
Forward KL, weight 0.1	3.4047	3.3355	3.2937	3.2641	3.2410	3.3078	36.4
Forward KL, weight 0.5	3.3920	3.3287	3.2931	3.2675	3.2482	3.3059	36.6
Forward KL, weight 1	3.3748	3.3184	3.2873	3.2653	3.2483	3.2988	36.1
Reverse KL, weight 0.5	3.3868	3.3261	3.2887	3.2632	3.2437	3.3017	35.9
Symmetric KL, weight 0.5	3.3875	3.3256	3.2901	3.2629	3.2435	3.3019	36.0
None (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5

(1) Selected sub-networks per step. Besides the default (All), which selects all five sub-networks at every step, we test three rules for the selected set $\mathcal{A}_\tau$ of Equation 8:

- *Sandwich:* Low and Max at every step and one of the other three uniformly at random;

Table 32: **Ablations (7)–(12) of SHELLELASTIC at the 119M–285M scale**, in the format of Table 31.

Setting	Val. loss ↓					5-shot acc. ↑	
	Low	Medium	High	Extra	Max	Avg.	Avg.
<i>(7) Normalization type</i>							
LayerNorm	3.4037	3.3379	3.2955	3.2655	3.2424	3.3090	35.8
RMSNorm (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(8) Ensembles of the five sub-networks</i>							
Learned per-token width router	3.7735	3.5611	3.4137	3.3055	3.2269	3.4562	35.1
Distillation from loss-weighted ensemble	3.3902	3.3275	3.2926	3.2673	3.2471	3.3049	37.0
Distillation from soft oracle	3.3982	3.3350	3.2980	3.2709	3.2501	3.3105	36.3
None (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(9) Stage-residual gradient scaling</i>							
$\gamma = 0$	3.4053	3.3740	3.3530	3.3348	3.3115	3.3557	36.1
$\gamma = 0.3$	3.3806	3.3317	3.2991	3.2762	3.2485	3.3072	36.6
$\gamma = 0.5$	3.3939	3.3360	3.2967	3.2700	3.2447	3.3083	36.0
$\gamma = 0.8$	3.4030	3.3365	3.2946	3.2658	3.2419	3.3084	37.0
$\gamma = 1$ (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(10) Trunk gradient of Low to Extra</i>							
Detached	3.9289	3.6481	3.4489	3.3134	3.2259	3.5130	35.2
Kept (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(11) Base width g and head dimension d_{head}</i>							
$g = 64, d_{\text{head}} = 64$	3.3999	3.3336	3.2919	3.2619	3.2383	3.3051	36.2
$g = 128, d_{\text{head}} = 64$	3.4021	3.3334	3.2929	3.2636	3.2400	3.3064	36.3
$g = 128, d_{\text{head}} = 128$ (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(12a) Gradient clipping (global norm)</i>							
0.5	3.4088	3.3392	3.2962	3.2655	3.2420	3.3104	36.3
1	3.4150	3.3431	3.2988	3.2679	3.2437	3.3137	35.9
2	3.4037	3.3342	3.2927	3.2633	3.2406	3.3069	36.6
4 (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(12b) AdamW ϵ</i>							
10^{-6}	3.4097	3.3411	3.3011	3.2737	3.2552	3.3162	35.7
10^{-8}	3.4055	3.3348	3.2920	3.2628	3.2398	3.3070	36.0
10^{-10}	3.4057	3.3373	3.2956	3.2660	3.2428	3.3095	36.4
10^{-14} (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5
<i>(12c) z-loss coefficient λ_z</i>							
10^{-3}	3.4096	3.3393	3.2970	3.2664	3.2414	3.3107	35.9
10^{-2}	3.4074	3.3363	3.2938	3.2640	3.2403	3.3084	36.5
10^{-4} (default)	3.4086	3.3375	3.2944	3.2641	3.2403	3.3090	36.5

- *Max + random*: Max and one of the other four;
- *Random*: one of the five sub-networks uniformly at random, as MatFormer and the fully elastic model do.

Observation: The loss of each sub-network follows how often it is trained, and All gives the lowest mean loss. Sandwich and Max + random keep the loss of Max but raise those of the sub-networks they select less often, by up to 0.024 and 0.086 nats, respectively, and Random raises every tier, Max by 0.095 nats.

(2) Loss weights. With all widths selected at every step, we vary the loss weights ω_s of Equation 8: the weight of Max rises from 1 to 2, 4 (the default) and 6, and the decaying weights (1, 0.5, 0.3, 0.2, 2) keep the relative weight of Max at 1/2 and double that of Low.

Observation: The loss weight of a width moves its loss in the same way as how often it is trained. Raising the weight of Max from 1 to 6 lowers its loss monotonically, by 0.015 nats, and raises the mean loss from 3.306 to 3.313, mostly through Low and Medium. The decaying weights match the

default at Max and lower the mean loss by 0.005 nats, mostly through Low. Training all widths at every step with weights (1, 1, 1, 1, 4) is thus within 0.005 nats of the best trade-off we found.

(3) Trunk normalization. We compare three trunk normalizations that all preserve the supported prefixes (Appendix B.2): group-local normalization over each $\mathcal{G}_k$ (the default), block-local normalization over every g coordinates, and a cumulative prefix norm, which normalizes each block of g coordinates with the statistics of all coordinates up to the end of that block. With $\mathcal{B}_b = \{(b-1)g + 1, \dots, bg\}$ the b -th block of g coordinates, all three apply Equation 3 with the statistics of a coordinate $j \in \mathcal{G}_k \cap \mathcal{B}_b$ taken over

$$\mathcal{G}_k \text{ (group-local), } \quad \mathcal{B}_b \text{ (block-local), } \quad \{1, \dots, bg\} \text{ (cumulative prefix).} \quad (25)$$

Observation: The three are within 0.001 nats of each other in mean loss and within 0.003 nats at every tier. They differ only in cost: the prefix norm, which needs cumulative statistics for every prefix of blocks, takes 6% more time per step and 23% more memory in our implementation, while block-local normalization costs the same as group-local normalization.

(4) Terminal norm. We compare normalizing all sg coordinates of sub-network s before its readout (the default, Equation 6) with normalizing each group separately, as in the trunk.

Observation: The default is better: 0.003 nats lower in mean loss, mostly at Low to High, and about 1 point higher in mean 5-shot accuracy.

(5) Readout. We compare separate readouts U_s for every width (the default) with one readout U shared by all widths, of which width s uses the first sg columns (with the split of Equation 23).

Observation: Separate readouts are better at every tier, by 0.006 nats in mean loss and 0.9 points in mean 5-shot accuracy.

(6) In-place distillation. We distill Low to Extra from Max by adding to the loss of each a weighted KL divergence between its output distribution and that of Max over the full vocabulary: forward KL with weights 0.1, 0.5 and 1, and reverse and symmetric KL with weight 0.5.

Observation: Distillation lowers the loss of Low to High, by up to 0.034 nats at Low, but raises that of Max, by up to 0.008 nats. The mean loss falls by up to 0.010 nats, but the mean 5-shot accuracy stays within 0.6 points of the default and a step takes 2.2 to 2.6 times as long, so we do not use it.

(7) Normalization type. We compare RMSNorm (the default) with LayerNorm with a gain and no bias.

Observation: Both give the same mean loss (3.309) and differ by at most 0.005 nats at any tier, so we keep RMSNorm.

(8) Ensembles of the sub-networks. One trunk pass yields the output distributions $p_s = \text{softmax}(z_s)$ of all five sub-networks, so ensembling them costs little beyond the readouts, and nested sub-networks may still err on different tokens. We use per-token ensembles $\sum_s \pi_s p_s$, whose weights π_s are non-negative and sum to one, in three ways:

- *Four ensembles of a trained supernet:* uniform weights, $\pi_s = 1/5$; confidence weights, $\pi_s \propto \max_v p_s(v)$; the most confident readout, which puts all weight on the sub-network with the largest $\max_v p_s(v)$; and the best fixed ensemble, whose weights minimize the loss of the tokens it is evaluated on. As an upper bound, an oracle puts all weight on the readout that assigns the target the highest probability.
- *Learned router:* a two-layer MLP ($1024 \rightarrow 256 \rightarrow 5$, SiLU) maps the last-layer hidden state of Max, after its terminal norm, to one logit per sub-network, and a softmax turns the five logits into the per-token weights π_s ; its output layer is initialized to zero, so the weights start uniform. It is trained jointly with the supernet, whose loss becomes the π -weighted sum of the five losses at every token plus 0.01 times their unweighted mean, which keeps every sub-network trained.

- *Distillation from ensembles*: every sub-network, including Max, adds a forward KL divergence with weight 0.5 toward one of two teacher ensembles, detached from the gradient: the *loss-weighted ensemble*, whose weights are the loss weights ω_s of Equation 8 normalized to sum to one (1/2 on Max, 1/8 on each other sub-network), or the *soft oracle*, $\pi_s \propto p_s(y)^\beta$ for target y , a smooth version of the oracle that favors the sub-networks assigning y a high probability, with β set at every step so that the weights have a mean normalized entropy of 0.7.

Observation: No ensemble beats Max alone (Table 33). The best fixed ensemble matches it, uniform and confidence weights are 0.03 nats worse, and the most confident readout is 0.068 nats worse; only the oracle is better, but it needs the target. The router collapses onto Max, which improves by 0.013 nats, while the other sub-networks, trained only through the 0.01 term, lose 0.041 to 0.365 nats. As predictors, the loss-weighted ensemble is 0.012 nats worse than Max and the soft oracle 0.143 nats better, as it uses the target; yet distillation toward them raises the loss of Max by 0.007 and 0.010 nats and lowers that of Low by 0.018 and 0.010 nats, as in-place distillation does in (6), and a step takes about 2.6 and 3.0 times as long.

Nested sub-networks versus separately trained models (Table 34, bottom):

- *Separately trained models ensemble well*. Averaging the Max of two supernet trained with the same settings lowers the loss by 0.056 nats, and that of three by 0.077 nats; uniform weights are already the best fixed weights, as the members are equally strong. The oracle over two such Max sub-networks also gains more than that over the five sub-networks, 0.23 against 0.19 nats. For the five sub-networks of one supernet, no fixed weights improve on Max: they share the trunk and are much more alike, and the narrower ones are weaker, so averaging only dilutes Max.
- *Low gives the largest share of the gain*. With Low and Max alone, the oracle is already 0.12 nats below Max, about two thirds of its gain with all five tiers.
- *Cost*. Each separate model needs its own training run and trunk pass, whereas one trunk pass yields all five sub-networks.

Outlook: Ensembling the sub-networks remains promising for elastic models: the oracle is 0.19 nats below Max alone and picks a narrower sub-network for 71% of held-out tokens (Table 34), and a label-free selection better than confidence could turn part of it into a gain at almost no extra cost.

Table 33: **Ensembles of the five sub-networks as predictors and as teachers**. Train and validation loss minus those of Max. The row labeled Router is the ensemble weighted per token by the learned router, which is trained together with its own supernet, so this row subtracts the losses of that supernet’s Max. The loss-weighted ensemble and the soft oracle are the two ensembles used as distillation teachers in ablation (8).

Ensemble	Loss – Max loss	
	Train loss	Val. loss
<i>Predictors</i>		
Uniform weights	+0.0313	+0.0316
Confidence weights	+0.0295	+0.0296
Most confident readout	+0.0648	+0.0678
Best fixed ensemble	0.0000	−0.0003
Router	0.0000	0.0000
<i>Teachers</i>		
Loss-weighted ensemble	–	+0.0116
Soft oracle (uses the target)	–	−0.1433
Oracle (uses the target)	−0.1731	−0.1851

(9) Stage-residual gradient scaling. The logits of width s_k are a sum of one term per group,

$$z_{s_k} = \alpha_{s_k} \sum_{j \leq k} h_{\mathcal{G}_j}^{(k)}(U_{s_k})_{:, \mathcal{G}_j}^\top, \quad h^{(k)} = N_{s_k}^{\text{out}}(x_{:, s_k g}^{(L)}), \quad (26)$$

Table 34: **The oracle on held-out C4 text** (final checkpoints). Top: share of tokens on which each tier assigns the target the highest probability (the oracle’s pick), its contribution to the oracle’s mean gain over Max (the four sum to 0.185), and the share of tokens on which it beats Max. Bottom: validation loss minus that of Max when the models in each row are ensembled with uniform weights, the best fixed weights or the oracle.

Tier	Low	Medium	High	Extra	Max
Picked by the oracle (%)	22.7	16.3	15.1	16.4	29.4
Contribution to the gain (nats)	0.0878	0.0454	0.0309	0.0210	–
Beats Max (%)	39.0	42.7	44.9	46.7	–

Members	Uniform	Best fixed	Oracle
Five tiers of one supernet	+0.0316	−0.0003	−0.1851
Low and Max of one supernet	+0.0355	−0.0002	−0.1187
Max of two supernets	−0.0564	−0.0564	−0.2306
Max of three supernets	−0.0767	−0.0767	−0.3401

so the supernet can be viewed as a sequence of stages: the core gives a base prediction, and each shell adds a correction to the terms of the groups before it, as in residual learning (He et al., 2016). To train the shells toward this role, we scale by γ the gradient that the loss of each width sends into the terms of the narrower groups, through the coordinates of the next narrower sub-network:

$$\frac{\partial \ell_{s_k}}{\partial h_{:s_{k-1}g}^{(k)}} \leftarrow \gamma \frac{\partial \ell_{s_k}}{\partial h_{:s_{k-1}g}^{(k)}}, \quad k = 2, \dots, K. \quad (27)$$

Every width has its own readout and terminal norm, which only its own loss trains, so for $\gamma > 0$ Adam, being invariant to the gradient scale, still trains them in full; the scaling only reduces the gradient that these coordinates pass on to the trunk. The forward pass is unchanged, and $\gamma = 1$ is the joint objective of Equation 8. With $\gamma = 0$, the loss of each width reaches the last-layer features of its own shell only, which then fit what the narrower groups leave unexplained, while the wider losses still train the narrower groups inside the trunk.

Observation: A γ between 0.3 and 0.8 lowers the loss of Low by up to 0.028 nats but raises those of Extra and Max by up to 0.012 nats and leaves the mean loss within 0.002 nats. With $\gamma = 0$, Low is 0.003 nats lower, but the wider widths lose more, from 0.036 nats at Medium to 0.071 at Extra and Max, and the mean loss rises by 0.047 nats, as the last-layer features of the narrower groups no longer adapt to the losses of the wider widths. We therefore keep $\gamma = 1$.

Outlook: Although $\gamma < 1$ does not lower the mean loss here, the stage view matches a property that **SHELLELASTIC** has but MatFormer and fully elastic models lack: it can grow after training. Since no group reads a later one (Equation 2) and every group is normalized on its own (Equation 3), a new shell $\mathcal{G}_{K+1}$ can be appended to a trained supernet and trained as a new stage, for example on downstream data, while all trained weights stay frozen, so every trained sub-network keeps its exact outputs. MatFormer and fully elastic models cannot grow this way: new FFN units of MatFormer write into the hidden states that its trained layers read, and new channels of a fully elastic model change the inputs of every trained weight in the wider sub-network, so their trained weights would operate on inputs they were not trained on.

(10) Trunk gradient of the narrower widths. We detach the trunk from the losses of Low to Extra, which then train only their terminal norms and readouts on a trunk trained by Max alone.

Observation: Detaching lowers the loss of Max by 0.014 nats but raises those of Low to Extra by 0.049 to 0.520 nats, so keeping their trunk gradients is much better overall.

(11) Base width and head dimension. At the same supported widths 512 to 1024 and the same parameter counts, halving the base width to $g = 64$ ($M = 16$, $\mathcal{S} = \{8, 10, 12, 14, 16\}$) keeps the 512-wide core and every shell 128 wide, so the groups, their normalization and the μ -**ELASTIC** factors ms are unchanged. We also halve the head dimension to $d_{\text{head}} = 64$, which $g = 64$ requires.

Observation: With $d_{\text{head}} = 64$ in both, $g = 64$ and $g = 128$ define the same model, and their mean losses differ by 0.001 nats. Halving the head dimension lowers the mean loss by 0.003 to 0.004 nats, mostly at Low. The head dimension thus has a small effect near run-to-run variation, and the base width none; we keep $g = d_{\text{head}} = 128$.

(12) Optimization HPs. We vary the gradient clipping (global norm 0.5, 1 and 2), the AdamW ϵ (10^{-6} , 10^{-8} and 10^{-10}) and the z -loss coefficient λ_z (10^{-3} and 10^{-2}) around the defaults of Table 9: global norm 4, $\epsilon = 10^{-14}$ and $\lambda_z = 10^{-4}$.

Observation: The defaults are within 0.002 nats in mean loss of clipping at 0.5 or 2, $\epsilon = 10^{-8}$ or 10^{-10} , and $\lambda_z = 10^{-3}$ or 10^{-2} . Clipping at 1 is 0.005 nats worse and $\epsilon = 10^{-6}$ is 0.007 nats worse, so the recipe is robust to these settings except for a large ϵ .